IBM DB2 10.1
for Linux, UNIX, and Windows

データ移動ユーティリティー: ガイドおよびリファレンス

2013 年 1 月更新版

SA88-4693-01
(英文原典：SC27-3869-01)

IBM DB2 10.1
for Linux, UNIX, and Windows

データ移動ユーティリティー: ガイドおよびリファレンス

2013 年 1 月更新版

SA88-4693-01
(英文原典：SC27-3869-01)

ご注意

本書および本書で紹介する製品をご使用になる前に、 321 ページの『付録 F. 特記事項』に記載されている情報をお読みください。

IBM 資料は、オンラインでご注文いただくことも、ご自分の国または地域の IBM 担当員を通してお求めいただくこともできます。

- オンラインで資料を注文するには、IBM Publications Center (http://www.ibm.com/shop/publications/order) をご利用ください。
- ご自分の国または地域の IBM 担当員を見つけるには、IBM Directory of Worldwide Contacts (http://www.ibm.com/planetwide/) をお調べください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典： SC27-3869-01
IBM DB2 10.1
for Linux, UNIX, and Windows
Data Movement Utilities Guide and
Reference
Updated January, 2013

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第1刷 2012.12

目次

本書について

本書では、以下の DB2® for Linux, UNIX, and Windows データ移動ユーティリティーについて説明し、それらの使用法を示しています。

- エクスポートおよびインポート

  エクスポートおよびインポート・ユーティリティーは、表またはビューと、他のデータベースまたは表計算プログラムとの間、DB2 データベースどうしの間、および DB2 データベースと DB2 Connect™ を使用するホスト・データベースとの間で、データの移動を実行します。エクスポート・ユーティリティーは、データベースのデータをオペレーティング・システム・ファイルに移動します。それらのファイルを使用して、エクスポートされたデータを別のデータベースにインポートまたはロードすることができます。

- ロード

  ロード・ユーティリティーは、表の中へのデータの移動、既存の索引の拡張、および統計の生成を行います。大量のデータを扱う場合は、インポート・ユーティリティーよりロード・ユーティリティーのほうがデータを高速で移動できます。エクスポート・ユーティリティーを使用してエクスポートされたデータは、ロード・ユーティリティーを使用してロードすることができます。

  ロード・ユーティリティーをパーティション・データベース環境で使用する場合、大量のデータを分散して、複数の異なるデータベース・パーティションにロードすることができます。

データ移動オプションの詳細なリストについては、 1 ページの『データ移動オプション』を参照してください。

データ移動のユーティリティーおよびリファレンス

データ移動オプション

DB2 for Linux, UNIX, and Windows で使用可能なデータ移動オプションにはさまざまなものがあります。このトピックでは、使用可能なデータ移動ツール、ユーティリティー、ストアード・プロシージャー、およびコマンドの概要を示します。

これらの表をガイドとして使用し、要件に合った最適なデータ移動オプションを判別する上で役立ててください。

表 1. ロード・ユーティリティー

方式	ロード・ユーティリティー
目的	新しく作成した表やすでにデータが入っている表に、大量のデータを効率よく移動します。
クロスプラットフォームの互換性	はい
最良事例の使用法	このユーティリティーは、パフォーマンスが最大の関心事である場合に最も適しています。このユーティリティーは、インポート・ユーティリティーの代わりに使用できます。これは、SQL の INSERTS を使用せず、フォーマット設定されたページを直接データベースに書き込むため、インポート・ユーティリティーよりも高速です。またロード・ユーティリティーには、データをログに記録しないオプションや、ロードされたデータのコピーを保管するために COPY オプションを使用するオプションがあります。ロード操作では、SMP および MPP 環境の CPU やメモリーなどのリソースを、十分に活用することができます。
参照	データのロード

表 2. INGEST ユーティリティー

方式	INGEST ユーティリティー
目的	ファイルおよびパイプからのデータを DB2 ターゲット表に流し込みます。その際、ターゲット表は使用可能な状態を保持します。
クロスプラットフォームの互換性	はい
最良事例の使用法	このユーティリティーは、パフォーマンスと可用性とのバランスがよく取れています。ただし、可用性のほうが重要な場合は、ロード・ユーティリティーの代わりに INGEST ユーティリティーを選択してください。インポート・ユーティリティーと同様に、INGEST はターゲット表が更新可能ビュー、範囲がクラスター化された表、またはニックネームである場合に適しています。ただし、INGEST ユーティリティーのほうがパフォーマンスの点で優れています。
参照	データの INGEST

表 3. インポート・ユーティリティー

方式	インポート・ユーティリティー
目的	外部ファイルから表、階層、ビュー、またはニックネームにデータを挿入します。
クロスプラットフォームの互換性	はい
最良事例の使用法	インポート・ユーティリティーは、以下の状態では、ロード・ユーティリティーの代わりに使用することができます。 • ターゲット表がビューである • ターゲット表に制約があり、ターゲット表を SET INTEGRITY ペンディング状態にしない • ターゲット表にトリガーがあり、それらを起動したい
参照	データのインポート

表 4. エクスポート・ユーティリティー

方式	エクスポート・ユーティリティー
目的	データベースから、いくつかある外部ファイル・フォーマットのいずれかにデータをエクスポートします。そうすると、後でデータをインポートまたはロードできます。
クロスプラットフォームの互換性	はい
最良事例の使用法	このユーティリティーは、データをさらに処理するかデータを別の表に移動するために外部ファイルにデータを保管する場合に、最も適しています。代わりに High Performance Unload (HPU) を使用できますが、別個に購入する必要があります。エクスポートは XML 列をサポートします。
参照	データのエクスポート

表 5. db2move コマンド

方式	**db2move** コマンド
目的	**db2move** ユーティリティーを COPY オプションを設定して使用すると、スキーマ・テンプレート (データが入っていてもいなくても) をソース・データベースからターゲット・データベースにコピーしたり、スキーマ全体をソース・データベースからターゲット・データベースに移動することができます。 **db2move** ユーティリティーに IMPORT または EXPORT オプションを設定して使用すると、DB2 データベース間で大量の表を移動できるようになります。
クロスプラットフォームの互換性	はい

表 5. db2move コマンド (続き)

最良事例の使用法	COPY オプションを設定して使用する場合、ソースとターゲットのデータベースは異なっていなければなりません。 COPY オプションは、スキーマ・テンプレートの作成時に便利です。クロスプラットフォームのバックアップおよびリストア操作のサポートがない場合には、データベースのクローン作成に IMPORT または EXPORT オプションを使用します。 IMPORT および EXPORT オプションは、**db2look** コマンドと一緒に使用します。
参照	• 「データベース: 管理の概念および構成リファレンス」の『スキーマのコピー』 • インポート表の再作成

表 6. RESTORE コマンド

方式	REDIRECT オプションおよび GENERATE SCRIPT オプションを指定した **RESTORE** コマンド
目的	既存のバックアップ・イメージにあるスクリプトを使用して、データベース全体を 1 つのシステムから別のシステムにコピーします。
クロスプラットフォームの互換性	制限があります。『参照』をご覧ください。
最良事例の使用法	このユーティリティーは、バックアップ・イメージが存在する場合に最も適しています。
参照	• 「データ・リカバリーと高可用性 ガイドおよびリファレンス」の『自動生成スクリプトを使用したリダイレクト・リストアの実行』 • 「データ・リカバリーと高可用性 ガイドおよびリファレンス」の『異なるオペレーティング・システムおよびハードウェア・プラットフォーム間のバックアップおよびリストア操作』

表 7. db2relocatedb コマンド

方式	**db2relocatedb** コマンド
目的	データベースを名前変更したり、1 つのデータベースまたはデータベースの一部を同じシステムまたは別のシステムに再配置します。
クロスプラットフォームの互換性	いいえ
最良事例の使用法	• このユーティリティーは、バックアップとリストアに時間がかかる場合に使用できます。 • このユーティリティーは、バックアップとリストアを使用してデータベースのコピーを移動または作成する代わりに使用できます。 • これは、テストなどの代替環境のためにデータベースのクローン作成を行うための簡単な方法としても使用されます。 • これは、表スペース・コンテナーをストレージ・デバイスの新しいセットに移動するために使用できます。

表 7. db2relocatedb コマンド (続き)

参照	「コマンド・リファレンス」の『db2relocatedb - データベースの再配置コマンド』

表 8. ADMIN_COPY_SCHEMA プロシージャー

方式	ADMIN_COPY_SCHEMA プロシージャー
目的	1 つのスキーマ内のすべてのオブジェクトのコピーを作成し、それらのオブジェクトを新しいスキーマに再作成できるようにします。このコピー操作は、データベース内のデータの有無に関わらず実行できます。
クロスプラットフォームの互換性	はい
最良事例の使用法	このユーティリティーは、スキーマ・テンプレートの作成時に便利です。これは、ソース・スキーマの動作に影響を与えずにスキーマを試す場合 (例えば、新規索引をテストする場合) にも便利です。 ADMIN_COPY_SCHEMA プロシージャーと **db2move** ユーティリティーの主な違いには次のものがあります。 • ADMIN_COPY_SCHEMA プロシージャーは単一のデータベース上で使用されますが、**db2move** ユーティリティーは複数のデータベース間で使用されます。 • **db2move** ユーティリティーは、表や索引などの物理オブジェクトを作成できない場合に呼び出されると、失敗します。 ADMIN_COPY_SCHEMA プロシージャーは、エラーをログに記録して続行します。 • ADMIN_COPY_SCHEMA プロシージャーは、「カーソルからロード」を使用してデータを 1 つのスキーマから別のスキーマに移動します。 **db2move** ユーティリティーは、「カーソルからロード」に似たリモート・ロードを使用します。これはデータをソース・データベースからプルします。
参照	「データベース: *管理の概念および構成リファレンス*」の『スキーマのコピー』

表 9. ADMIN_MOVE_TABLE プロシージャー

方式	ADMIN_MOVE_TABLE プロシージャー
目的	データはオンライン状態でアクセス可能なまま、表内のデータを、同じ名前の新しい表オブジェクト (ただし、ストレージ特性は異なる可能性がある) へ移動できるようにします。
クロスプラットフォームの互換性	はい

表 9. *ADMIN_MOVE_TABLE プロシージャー (続き)*

最良事例の使用法	このユーティリティーは、表データを新しい表オブジェクトに移動するプロセスを自動化します。このとき、データはオンラインのままになり、選択、挿入、更新、および削除のためのアクセスができます。また、表が移動されるときに、コンプレッション・ディクショナリーを生成することもできます。 • 同一表スペースに複数の移動を同時に行うことは避けてください。 • 表に対するアクティビティーが低いときに、このプロシージャーを実行してください。 • 複数ステップの移動操作を使用してください。INIT フェーズと COPY フェーズはいつでも呼び出すことができます。ステージング表のサイズを小さくしておくためには、REPLAY フェーズを複数回実行します。その後、表に対するアクティビティーが低いときに SWAP を発行してください。 • ユニーク索引のない表や、索引のない表を処理する場合は、オフラインの表移動の使用を検討してください。
参照	• 「コマンド・リファレンス」の『ADMIN_MOVE_TABLE プロシージャー - オンライン表の移動』 • ADMIN_MOVE_TABLE プロシージャーによるオンラインでの表の移動

表 10. *スプリット・ミラー*

方式	スプリット・ミラー
目的	クローン・データベース、スタンバイ・データベース、またはバックアップ・データベースを作成します。
クロスプラットフォームの互換性	いいえ
最良事例の使用法	• 主データベースで障害が発生した場合のダウン時間を短縮するためにスタンバイ・システムを作成します。 • バックアップ操作を、有効な実動マシンから切り離して分割データベースに移動します。 • テストなどの代替環境のためにデータベースのクローン作成を行うための簡単な方法として使用されます。
考慮事項	• データベースの分割バージョンでは、DMS 表スペースのみをバックアップできます。 • 通常、ストレージ・システムに搭載されている何らかのフラッシュ・コピー・テクノロジーと一緒に使用されます。 • 代わりに、データベースが中断した時にファイルのコピーを実行するという方法がありますが、この場合、データベースのストレージの量が倍になります。
参照	「*データ・リカバリーと高可用性 ガイドおよびリファレンス*」の『オンライン・スプリット・ミラーおよびサスペンド入出力サポートによる高可用性』

エクスポート・ユーティリティー

エクスポート・ユーティリティーの概要

エクスポート・ユーティリティーは、SQL の SELECT ステートメントまたは XQuery ステートメントを使用してデータを抽出し、その情報をファイルに格納するためのユーティリティーです。出力ファイルを使用すれば、後にインポートやロードを利用してデータの移動をすることも、分析のためにデータをアクセス可能な状態にしておくこともできます。

エクスポート・ユーティリティーは、シンプルで柔軟性の高いデータ移動ユーティリティーです。このユーティリティーを実行するには、CLP で **EXPORT** コマンドを実行するか、ADMIN_CMD ストアード・プロシージャーを呼び出すか、ユーザー・アプリケーションで db2Export API を呼び出します。

基本的なエクスポート操作で必須の項目は、以下のとおりです。

- エクスポートしたデータを格納するためのオペレーティング・システム・ファイルのパスと名前
- 入力ファイル内のデータのフォーマット
  エクスポート操作の出力ファイルのデータ・フォーマットとしては、IXF と DEL がサポートされています。
- エクスポートするデータの指定
  ほとんどのエクスポート操作では、エクスポートのために取得するデータを指定した SELECT ステートメントを指定する必要があります。型付き表をエクスポートするときには、SELECT ステートメントを明示的に実行する必要はありません。指定しなければならないのは、階層内の副表のトラバース順序だけです。

IXF 形式のデータを移動する必要がある場合は、DB2 Connect でエクスポート・ユーティリティーを使用できます。

追加のオプション

エクスポート操作をカスタマイズするために使用できるパラメーターがいくつかあります。ファイル・タイプ修飾子には、データのフォーマット、日付と時刻のスタンプ、コード・ページを変更するためのオプションや、特定のデータ・タイプを別個のファイルに書き出すためのオプションなどが用意されています。**METHOD** パラメーターを使用すれば、エクスポート・データで使用する列名を変更して指定できます。

XML データ・タイプの列が 1 つ以上含まれている表からデータをエクスポートすることも可能です。エクスポート文書を格納する方法の詳細を指定するには、**XMLFILE**、**XML TO**、**XMLSAVESCHEMA** の各パラメーターを使用します。

エクスポート・ユーティリティーのパフォーマンスを改善する方法がいくつかあります。エクスポート・ユーティリティーは、組み込み SQL アプリケーションであり、内部で SQL フェッチを実行するので、SQL 操作に当てはまる最適化の手法は、エクスポート・ユーティリティーにも当てはまります。例えば、大きなバッファー・プール、索引作成、ソート・ヒープなどを活用できます。さらに、出力ファイルに関するデバイスの競合を最小化する

ために、コンテナーやロギング用デバイスとは異なるデバイスに出力ファイルを配置する、という方法もあります。

メッセージ・ファイル
エクスポート・ユーティリティーは、エラー・メッセージ、警告メッセージ、通知メッセージを標準の ASCII テキスト・メッセージ・ファイルに書き込みます。CLP 以外のすべてのインターフェースでは、そのファイルの名前を事前に **MESSAGES** パラメーターで指定しておく必要があります。CLP を使用する場合にメッセージ・ファイルを指定しなければ、エクスポート・ユーティリティーは、メッセージを標準出力に書き込みます。

IBM® Data Studio バージョン 3.1 以降では、次のタスクのためにタスク・アシスタントを使用できます: データのエクスポート. タスク・アシスタントは、オプションの設定、タスク実行のために自動生成されたコマンドの確認、およびそれらのコマンドの実行のプロセスをガイドします。詳しくは、タスク・アシストを使用したデータベースの管理を参照してください。

エクスポート・ユーティリティーを使用するために必要な特権と権限

特権は、データベース・リソースの作成、更新、削除、アクセスを可能にするためのものです。権限レベルは、データベース・マネージャーの保守とユーティリティーの高水準操作に各種の特権を対応付けるための手段になります。

特権と権限を組み合わせて、データベース・マネージャーとデータベース・オブジェクトに対するアクセスを制御します。アクセスできるのは、適切な許可 (必要な特権または権限) が与えられているオブジェクトだけになります。

DATAACCESS 権限か、エクスポート操作にかかわっている各表またはビューに対する CONTROL 特権または SELECT 特権が必要です。

LBAC で保護されたデータをエクスポートする場合は、セッション許可 ID に、エクスポートする行または列に対する読み取り許可が必要です。保護された行の読み取り許可を持たないセッション許可 ID では、その行をエクスポートできません。保護された列が SELECT ステートメントに含まれていて、セッション許可 ID にその列の読み取り許可がなければ、エクスポート・ユーティリティーは失敗し、エラー (SQLSTATE 42512) が返されます。

データのエクスポート

エクスポート・ユーティリティーを使用して、データベースからファイルにデータをエクスポートできます。そのファイルは、いずれかの外部ファイル形式になります。SQL SELECT ステートメントを提供するか、または型付き表の階層情報を提供することにより、エクスポートするデータを指定できます。

始める前に

データベースからデータをエクスポートするには、DATAACCESS 権限、CONTROL 特権、操作に関わっているそれぞれの表またはビューに関する SELECT 特権のいずれかが必要です。

エクスポート・ユーティリティーを実行するには、まずデータのエクスポート元になるデータベースに接続している (または、暗黙接続が可能な状態になっている) 必要があります。暗黙的な接続が可能である場合には、デフォルトのデータベースへの接続が確立されます。 Linux、UNIX、または Windows クライアントから Linux、UNIX、または Windows データベース・サーバーへのユーティリティーのアクセスは、DB2 Connect ゲートウェイまたはループバック環境を介してではなく、エンジンからの直接接続でなければなりません。

このユーティリティーは COMMIT ステートメントを実行するので、エクスポート・ユーティリティーの呼び出し前に COMMIT ステートメントまたは ROLLBACK ステートメントを実行して、すべてのトランザクションを完了し、すべてのロックを解放しておいてください。アプリケーションが表にアクセスし、かつ切断するのに別々の接続を使用することに関する要件はありません。

構造化タイプ列を持つ表はエクスポートできません。

手順

エクスポート・ユーティリティーを実行するには、次のようにします。

- コマンド行プロセッサー (CLP) で **EXPORT** コマンドを指定します。
- db2Export アプリケーション・プログラミング・インターフェース (API) を呼び出します。
- **EXPORT** コマンドに関する IBM Data Studio のタスク・アシストを開きます。

例

シンプルなエクスポート操作の場合に指定する必要があるのは、SELECT ステートメントのターゲット・ファイル、ファイル形式、ソース・ファイルだけです。

例えば、以下のようにします。

```
db2 export to filename of ixf select * from table
```

filename は、作成/エクスポートする出力ファイルの名前、`ixf` はファイル形式、*table* は、コピーするデータが含まれている表の名前です。

ただし、警告メッセージやエラー・メッセージを書き込むメッセージ・ファイルを指定することも可能です。そのためには、**MESSAGES** パラメーターとメッセージ・ファイル名 (この場合は `msg.txt`) を追加します。例えば、以下のようにします。

```
db2 export to filename of ixf messages msgs.txt select * from table
```

XML データのエクスポート

XML データをエクスポートする際、結果として作成される QDM (XQuery データ・モデル) インスタンスは、エクスポートされるリレーショナル・データを含むメイン・データ・ファイルとは別のファイル (1 つまたは複数) に書き込まれます。XMLFILE オプションおよび XML TO オプションのどちらも指定されていない場合でも、そのようになります。

デフォルトで、エクスポートされた QDM インスタンスはすべて同じ XML ファイルに連結されます。 `XMLINSEPFILES` ファイル・タイプ修飾子を使用して、各 QDM インスタンスが別のファイルに書き込まれるように指定できます。

ただし XML データは、メイン・データ・ファイル内で XML データ指定子 (XDS) によって表されます。 XDS は XDS という名前の XML タグによって表されるストリングであり、列内の実際の XML データについての情報を記述する属性が付随しています。そのような情報には、実際の XML データが含まれているファイルの名前、およびそのファイル内の XML データのオフセットおよび長さが含まれます。

エクスポートされた XML ファイルの宛先パスおよびベース名は、XML TO および XMLFILE オプションを使用して指定できます。 XML TO または XMLFILE オプションが指定されている場合、エクスポートされた XML ファイルの名前として XDS の FIL 属性に格納される名前の形式は xmlfilespec.xxx.xml です (xmlfilespec は XMLFILE オプションに指定された値、xxx はエクスポート・ユーティリティーによって生成された XML ファイルの順序番号)。そうでない場合、エクスポートされた XML ファイル名の形式は exportfilename.xxx.xml となります (exportfilename は EXPORT コマンドのために指定されるエクスポートされた出力ファイルの名前、xxx はエクスポート・ユーティリティーによって生成された XML ファイルの順序番号)。

デフォルトで、エクスポートされた XML ファイルはエクスポートされたデータ・ファイルのパスに書き込まれます。エクスポートされた XML ファイルのデフォルトのベース名は、エクスポートされたデータ・ファイル名に 3 桁の順序番号、および .xml 拡張子を付加したものです。

例

以下の例では、4 つの列および 2 つの行を含む表 USER.T1 を想定します。

```
C1 INTEGER
C2 XML
C3 VARCHAR(10)
C4 XML
```

表 11. USER.T1

C1	C2	C3	C4
2	<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to><from> Me</from><heading>note1</heading> <body>Hello World!</body></note>	'char1'	<?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00"><to>Him</ to><from> Her</from><heading>note2</ heading>< body>Hello World!</body></note>
4	NULL	'char2'	?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00">to>Us</ to><from> Them</ from><heading>note3</heading> <body>Hello World!</body></note>

例 1

以下のコマンドは、USER.T1 の内容を Delimited ASCII (DEL) 形式でファイル "/mypath/t1export.del" にエクスポートします。 XML TO および XMLFILE オプションが指定されていないので、列 C2 および C4 に含まれる XML 文書はメインのエクスポート・ファイル "/mypath" と同じパスに書き込まれます。これらのファイ

ルのベース名は、"t1export.del.xml" です。 XMLSAVESCHEMA オプションは、XML スキーマ情報がエクスポート手順の実行中に保存されることを示します。

```
EXPORT TO /mypath/t1export.del OF DEL XMLSAVESCHEMA SELECT * FROM USER.T1
```

エクスポート・ファイル "/mypath/t1export.del" には、以下が含まれます。

```
2,"<XDS FIL='t1export.del.001.xml' OFF='0' LEN='144' />","char1",
"<XDS FIL='t1export.del.001.xml' OFF='144' LEN='145' />"
4,,"char2","<XDS FIL='t1export.del.001.xml' OFF='289'
LEN='145' SCH='S1.SCHEMA_A' />"
```

エクスポートされた XML ファイル "/mypath/t1export.del.001.xml" には、以下が含まれます。

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
   <from>Me</from><heading>note1</heading><body>Hello World!</body>
   </note><?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00"><to>Him
   </to><from>Her</from><heading>note2</heading><body>Hello World!
   </body></note><?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00">
   <to>Us</to><from>Them</from>heading>note3</heading><body>
   Hello World!</body></note>
```

例 2

以下のコマンドは、USER.T1 の内容を DEL 形式でファイル "t1export.del" にエクスポートします。列 C2 および C4 に含まれる XML 文書は、パス "/home/user/xmlpath" に書き込まれます。 XML ファイルはベース名 "xmldocs" を使用して名前が付けられ、複数のエクスポートされた XML 文書が同じ XML ファイルに書き込まれます。 XMLSAVESCHEMA オプションは、XML スキーマ情報がエクスポート手順の実行中に保存されることを示します。

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs XMLSAVESCHEMA SELECT * FROM USER.T1
```

エクスポートされた DEL ファイル "/home/user/t1export.del" には、以下が含まれます。

```
2,"<XDS FIL='xmldocs.001.xml' OFF='0' LEN='144' />","char1",
"<XDS FIL='xmldocs.001.xml' OFF='144' LEN='145' />"
4,,"char2","<XDS FIL='xmldocs.001.xml' OFF='289'
LEN='145' SCH='S1.SCHEMA_A' />"
```

エクスポートされた XML ファイル "/home/user/xmlpath/xmldocs.001.xml" には、以下のものが含まれます。

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
   <from>Me</from><heading>note1</heading><body>Hello World!</body>
   </note><?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00">
   <to>Him</to><from>Her</from><heading>note2</heading><body>
   Hello World!</body></note><?xml version="1.0" encoding="UTF-8" ?>
   <note time="14:00:00"><to>Us</to><from>Them</from><heading>
   note3</heading><body>Hello World!</body></note>
```

例 3

以下のコマンドは例 2 と似ていますが、それぞれのエクスポートされた XML 文書が別の XML ファイルに書き込まれることが異なります。

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs MODIFIED BY XMLINSEPFILES XMLSAVESCHEMA
SELECT * FROM USER.T1
```

エクスポート・ファイル "/mypath/t1export.del" には、以下が含まれます。

```
2,"<XDS FIL='xmldocs.001.xml' />","char1","XDS FIL='xmldocs.002.xml' />"
4,,"char2","<XDS FIL='xmldocs.004.xml' SCH='S1.SCHEMA_A' />"
```

エクスポートされた XML ファイル "/home/user/xmlpath/xmldocs.001.xml" には、以下のものが含まれます。

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
   <from>Me</from><heading>note1</heading><body>Hello World!</body>
   </note>
```

エクスポートされた XML ファイル "/home/user/xmlpath/xmldocs.002.xml" には、以下のものが含まれます。

```
?xml version="1.0" encoding="UTF-8" ?>note time="13:00:00">to>Him/to>
   from>Her/from>heading>note2/heading>body>Hello World!/body>
   /note>
```

エクスポートされた XML ファイル "/home/user/xmlpath/xmldocs.004.xml" には、以下のものが含まれます。

```
<?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00"><to>Us</to>
   <from>Them</from><heading>note3</heading><body>Hello World!</body>
   </note>
```

例 4

次のコマンドは、 XQuery の結果を XML ファイルに書き込みます。

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs MODIFIED BY XMLNODECLARATION select
xmlquery( '$m/note/from/text()' passing by ref c4 as "m" returning sequence)
   from USER.T1
```

エクスポートされた DEL ファイル "/mypath/t1export.del" には、以下が含まれます。

```
"<XDS FIL='xmldocs.001.xml' OFF='0' LEN='3' />"
"<XDS FIL='xmldocs.001.xml' OFF='3' LEN='4' />"
```

エクスポートされた XML ファイル "/home/user/xmlpath/xmldocs.001.xml" には、以下のものが含まれます。

```
HerThem
```

注: 特定の XQuery の結果は、整形式 XML 文書を生成しません。そのため、この例でエクスポートされたファイルを XML 列に直接インポートすることはできません。

エクスポート・セッション - CLP の例

例 1

次の例は、SAMPLE データベースの中の STAFF 表から myfile.ixf へ、IXF フォーマットの出力で情報をエクスポートする方法を示しています。ユーザーはすでにデータベースに接続していることが必要です。データベース接続が DB2 Connect を介していない場合、索引定義が存在するならそれは出力ファイルに格納されます。そうでなければ、データだけが格納されます。

```
db2 export to myfile.ixf of ixf messages msgs.txt select * from staff
```

例 2
次の例は、SAMPLE データベースの中の STAFF 表から awards.ixf へ、部署 (dept) 20 の従業員に関する情報を IXF フォーマットの出力でエクスポートする方法を示しています。ユーザーはすでにデータベースに接続していることが必要です。

```
db2 export to awards.ixf of ixf messages msgs.txt select * from staff
   where dept = 20
```

例 3
次の例は LOB を DEL ファイルにエクスポートする方法を示しています。

```
db2 export to myfile.del of del lobs to mylobs/
   lobfile lobs1, lobs2 modified by lobsinfile
   select * from emp_photo
```

例 4
次の例は LOB を DEL ファイルにエクスポートする方法を示しています。ここでは、最初のディレクトリーにファイルを入れることができない場合のために 2 番目のディレクトリーを指定しています。

```
db2 export to myfile.del of del
   lobs to /db2exp1/, /db2exp2/ modified by lobsinfile
   select * from emp_photo
```

例 5
次の例はデータを DEL ファイルにエクスポートする方法を示しています。ここでは、単一引用符をストリング区切り文字として使用し、セミコロンを列の区切り文字として使用し、コンマを小数点として使用します。データを再びデータベースにインポートする場合、これと同じ規則を使用する必要があります。

```
db2 export to myfile.del of del
   modified by chardel'' coldel; decpt,
   select * from staff
```

LBAC で保護されたデータのエクスポートに関する考慮事項

LBAC (ラベル・ベースのアクセス制御) で保護されたデータをエクスポートするときに、実際にエクスポートされるのは、LBAC 信用証明情報に読み取り権限があるデータに限られます。

LBAC 信用証明情報に行の読み取り権限がなければ、その行はエクスポートされませんが、エラーは返されません。LBAC 信用証明情報に列の読み取り権限がなければ、エクスポート・ユーティリティーは失敗し、エラー (SQLSTATE 42512) が返されます。

データ・タイプ DB2SECURITYLABEL の列の値は、区切り文字で囲まれた生データとしてエクスポートされます。元のデータに区切り文字が含まれていれば、区切り文字が二重になります。エクスポート値を構成するバイトにそれ以外の変更は加えられません。したがって、DB2SECURITYLABEL データが含まれているデータ・ファイルには、改行や用紙送りなどの出力不能の ASCII 文字が組み込まれる可能性があります。

データ・タイプ DB2SECURITYLABEL の列の値を理解できる形式でエクスポートするには、SELECT ステートメントで SECLABEL_TO_CHAR スカラー関数を使用して、その値をセキュリティー・ラベル・ストリング・フォーマットに変換できます。

例

以下の例では、出力を DEL 形式にして、ファイル myfile.del に書き込みます。データのエクスポート元は、以下のステートメントで作成された REPS という名前の表です。

```
create table reps (row_label db2securitylabel,
id integer,
name char(30))
security policy data_access_policy
```

次に示すのは、row_label 列の値をデフォルト形式でエクスポートする例です。

```
db2 export to myfile.del of del select * from reps
```

このデータ・ファイルの内容を読もうとしても、ほとんどのテキスト・エディターでは、あまり読みやすくない状態で表示されます。row_label 列の値には、いくつかの ASCII 制御文字が含まれている可能性が高いからです。

次に示すのは、row_label 列の値をセキュリティー・ラベル・ストリング・フォーマットでエクスポートする例です。

```
db2 export to myfile.del of del select SECLABEL_TO_CHAR
(row_label,'DATA_ACCESS_POLICY'), id, name from reps
```

この例で作成したデータ・ファイルの抜粋を以下に示します。セキュリティー・ラベルの形式が読みやすくなっています。

```
...
"Secret:():Epsilon 37", 2005, "Susan Liu"
"Secret:():(Epsilon 37,Megaphone,Cloverleaf)", 2006, "Johnny Cogent"
"Secret:():(Megaphone,Cloverleaf)", 2007, "Ron Imron"
...
```

表のエクスポートに関する考慮事項

標準的なエクスポート操作では、既存の表に挿入されているかロードされているデータを選択的に出力します。ただし、表全体をエクスポートして、後からインポート・ユーティリティーによって表を再作成することも可能です。

表をエクスポートするには、PC/IXF ファイル形式を指定する必要があります。そのようにして保管した表を索引と一緒に再作成するには、インポート・ユーティリティーを CREATE モードで使用します。ただし、以下のような条件が存在する場合は、エクスポート IXF ファイルに一部の情報が保管されません。

- 索引列名に 16 進値 0x2B または 0x2D が含まれる
- 表に XML 列が含まれる。
- 表がマルチディメンション・クラスター (MDC) 表である。
- 表に表パーティション・キーが含まれる。
- コード・ページ変換が原因で索引名が 128 バイトより大きくなっている。
- 表が保護されている。

- **EXPORT** コマンドに SELECT * FROM *tablename* 以外のアクション・ストリングが含まれている。
- エクスポート・ユーティリティーで **METHOD N** パラメーターが指定されている。

失われる表属性のリストについては、「表のインポートに関する考慮事項」を参照してください。保管されない情報がある場合は、表の再作成時に警告 SQL27984W が返されます。

注: インポートの CREATE モードは、非推奨になります。表のキャプチャーと再作成には、**db2look** ユーティリティーを使用してください。

索引情報

索引で指定した列名に「-」か「+」のいずれかの文字が含まれる場合は、索引情報は収集されず、警告 SQL27984W が戻されます。エクスポート・ユーティリティーは処理を完了し、エクスポート・データに影響はありません。ただし、索引情報は IXF ファイルに保管されません。したがって、**db2look** ユーティリティーを使用して、索引を別途作成する必要があります。

スペースに関する制限

エクスポートするデータが、エクスポート作業を行う OS のファイル・システムの使用可能領域を超えると、エクスポート操作は失敗します。その場合は、WHERE 節で条件を指定することにより、選択されるデータの量を制限して、エクスポート先のファイル・システムにうまく収まるようにしてください。すべてのデータをエクスポートするために、エクスポート・ユーティリティーを複数回実行することができます。

他のファイル形式の表

エクスポート時に IXF ファイル形式を使用しない場合は、出力ファイルにレコード・データは組み込まれますが、ターゲット表の記述は組み込まれません。表とデータを再作成するには、ターゲット表を作成してから、ロード・ユーティリティーまたはインポート・ユーティリティーを使用してその表にデータを取り込みます。元の表定義をキャプチャーし、対応するデータ定義言語 (DDL) を生成するには、**db2look** ユーティリティーを使用します。

型付き表のエクスポートに関する考慮事項

DB2 エクスポート・ユーティリティーを使用すれば、後からインポートできる状態で型付き表のデータを移動できます。エクスポートでは、特定の順序を指定し、中間のフラット・ファイルを作成することによって、型付き表の階層構造間でデータを移動します。

エクスポート・ユーティリティーでは、型付き表を操作するときに、出力ファイルに格納するデータを制御するために、ターゲット表の名前だけを指定しますが、オプションとして WHERE 節も指定できます。副選択ステートメントも記述できますが、そのためには、ターゲット表の名前と WHERE 節だけを指定します。階層をエクスポートするときに、全選択や SELECT ステートメントを指定することはできません。

トラバース順序を使用した階層の保存

型付き表は、階層になっている場合も可能です。階層間でデータを移動するには、いくつかの方法があります。

- 1 つの階層からそれと同一の階層への移動
- 1 つの階層から、より大きな階層のサブセクションへの移動
- 大きな階層のサブセクションから別の階層への移動

階層内の型の識別方法はデータベースによって違います。つまり、同じ型でもデータベースが異なると ID が違います。そのため、それらのデータベース間でデータを移動する場合、データを正しく移動するために同じ型のマッピングを実行する必要があります。

型付き表で使用するマッピングのことを*トラバース順序*といいます。つまり、階層内のすべてのスーパー表と副表を上から下へ、左から右へ進む順序です。エクスポート操作中にそれぞれの型の行が書き出される前に、ID が索引値に変換されます。この索引値は、1 から、階層内の関連する型の数までの範囲のいずれかの数になります。階層内を特定の順序 (トラバース順序) で移動する場合、それぞれの型に番号を付けることによって索引値が生成されます。図 1 は、以下のような 4 つの有効なトラバース順序のある階層を示しています。

- Person、Employee、Manager、Architect、Student
- Person、Student、Employee、Manager、Architect
- Person、Employee、Architect、Manager、Student
- Person、Student、Employee、Architect、Manager

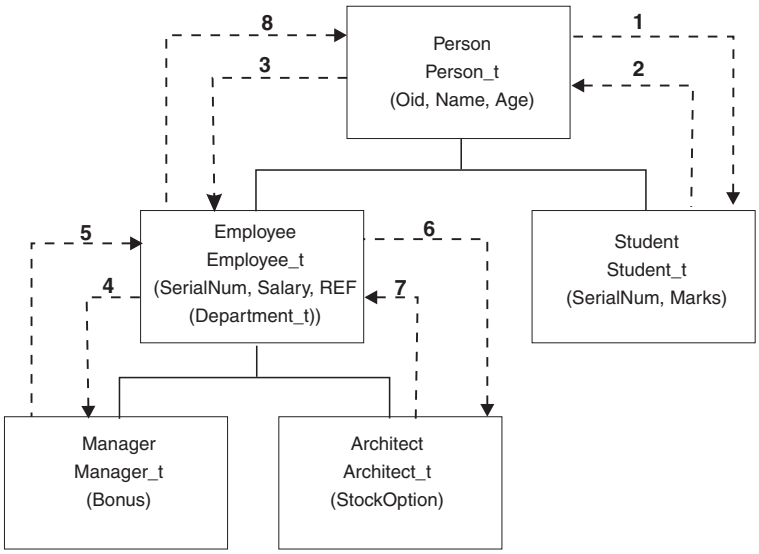


図 1. 階層の例

表階層間でデータを移動する場合、トラバース順序は他のデータとの相対関係でデータがどこに移動するかを決めるものとなるため重要です。トラバース順序には、*デフォルト*と*ユーザー指定*という 2 つのタイプがあります。

デフォルトのトラバース順序

デフォルトのトラバース順序では、関連するすべての型が、階層内の特定の開始点から到達可能な階層内の型をすべて参照します。デフォルト順序には階層内のすべての表が組み込まれており、それぞれの表の順序は OUTER 順序述部で使用されている方式に従ったものです。例えば、図 1 のデフォルトのトラバース順序は、点線で示されているとおり、
Person、Student、Employee、Manager、Architect になります。

ファイル・フォーマットが違うと、デフォルトのトラバース順序の動作が違います。 PC/IXF ファイル・フォーマットにデータをエクスポートすると、関連のあるすべての型、それらの定義、および関連している表に関するレコードが作成されます。また、エクスポート・ユーティリティーによって索引値からそれぞれの表へのマッピングが完了します。 PC/IXF ファイル・フォーマットで作業をする場合は、デフォルトのトラバース順序を使用してください。

ASC または DEL ファイル・フォーマットでは、ソースとターゲットの階層の構造が同じであっても、型付き行および型付き表の作成順が違う可能性があります。その場合、デフォルトのトラバース順序で階層を進むうちに時間の差が発生することになります。デフォルトのトラバース順序を使用する場合、ソースとターゲットのどちらにおいても、それぞれの型の作成日時が階層内を移動するためのトラバース順序を決定します。ソース階層およびターゲット階層の両方で、それぞれの型の作成順序が同じであること、およびソースとターゲットの構造が同じであることを確かめてください。この条件が満たされない場合は、ユーザー指定のトラバース順序を選択してください。

ユーザー指定のトラバース順序

ユーザー指定のトラバース順序では、使用する関連のある型をユーザーがトラバース順序リストで定義します。デフォルトのトラバース順序では、階層内のすべての表がエクスポートされますが、この順序では、階層をトラバースする方法とエクスポートする副表の概略を指定します。

トラバース順序を定義するときには、開始点と階層を下るパスを指定しますが、副表は *pre-order* 方式でトラバースする必要があります。つまり、階層内のそれぞれの分岐の最下位にまで達しないと、新しい分岐のトラバースを開始できません。エクスポート・ユーティリティーは、指定されたトラバース順序がこの条件に違反していないかどうかを検査します。その条件を満たすための 1 つの方法は、階層の最上部 (ルート表) から下の階層 (副表) に進んで最も低い副表に達してから、そのスーパー表に戻り、次の「右端の」副表まで達したら、またそのスーパー表のさらに上に戻って、その副表を下っていき、このような過程を続けていく、ということです。

階層のトラバース順序を制御する場合は、エクスポートおよびインポート・ユーティリティーで必ず同じトラバース順序が使用されるようにしてください。

例 1

以下の例は、図 1 の階層構造に基づいています。階層全体をエクスポートするには、以下のコマンドを入力します。

```
DB2 CONNECT TO Source_db
DB2 EXPORT TO entire_hierarchy.ixf OF IXF HIERARCHY STARTING Person
```

パラメーター **HIERARCHY STARTING** を Person に設定しているので、表 PERSON からデフォルトのトラバース順序が始まります。

例 2

階層全体をエクスポートするものの、20 歳以上の人のデータだけを抽出する場合は、以下のコマンドを入力します。

```
DB2 CONNECT TO Source_db
   DB2 EXPORT TO entire_hierarchy.del OF DEL HIERARCHY (Person,
      Employee, Manager, Architect, Student) WHERE Age>=20
```

パラメーター **HIERARCHY** を Person, Employee, Manager, Architect, Student に設定しているので、ユーザー指定のトラバース順序になっています。

ID 列のエクスポートに関する考慮事項

エクスポート・ユーティリティーでは、ID 列が含まれている表のデータをエクスポートできます。ただし、ID 列がある場合は、出力ファイルの形式に関する選択肢が限定されます。

エクスポート操作で指定する SELECT ステートメントが SELECT * FROM *tablename* という形式であり、METHOD オプションを使用しない場合は、ID 列のプロパティーを IXF ファイルにエクスポートできます。 **IMPORT** コマンドの REPLACE_CREATE オプションと CREATE オプションを使用すれば、ID 列のプロパティーを組み込んで表を再作成することが可能です。ただし、タイプ GENERATED ALWAYS の ID 列が含まれている表からエクスポート IXF ファイルを作成した場合は、インポート操作の実行時に identityignore ファイル・タイプ修飾子を指定しない限り、データ・ファイルを正常にインポートできません。そうでない場合は、すべての行がリジェクトされ、SQL3550W が生成されます。

注: IMPORT コマンドの CREATE オプションと REPLACE_CREATE オプションは非推奨で、将来のリリースでは廃止される可能性があります。

LOB のエクスポートに関する考慮事項

ラージ・オブジェクト (LOB) 列が含まれている表をエクスポートする場合は、1 つの LOB 値につき最大で 32 KB までのデータをエクスポートし、そのデータと残りの列データを同じファイルに格納する、というのがデフォルトのアクションになります。32 KB を超える LOB 値をエクスポートする場合は、切り捨てを避けるために、LOB データを別のファイルに書き込む必要があります。

LOB を独自のファイルに書き込むことを指定するには、lobsinfile ファイル・タイプ修飾子を使用します。この修飾子を指定すると、エクスポート・ユーティリティーは、LOBS TO 節で指定されているディレクトリーに LOB データを格納します。 LOBS TO または LOBFILE を使用すると、lobsinfile ファイル・タイプ修飾子が暗黙的にアクティブ化されます。デフォルトで、LOB 値は、エクスポート・リレーショナル・データが書き込まれるのと同じパスに書き込まれます。 LOBS TO オプションに 1 つ以上のパスが指定されていると、エクスポート・ユーティリティーは、各パスの間を循環し、正常な LOB 値をそれぞれ適切な LOB ファイルに書き込みます。さらに、LOBFILE オプションを使用して、出力 LOB ファイルの名前を指定することもできます。LOBFILE オプションが指定されている場合、lobfilename のフォーマットは lobfilespec.xxx.lob となります。ここで lobfilespec は

LOBFILE オプションに指定された値であり、xxx はエクスポート・ユーティリティーによって生成された LOB ファイルのシーケンス番号です。それ以外の場合、lobfilename は exportfilename.xxx.lob のフォーマットになります。ここで、exportfilename は EXPORT コマンドに指定されたエクスポートされる出力ファイルの名前であり、xxx はエクスポート・ユーティリティーによって生成された LOB ファイルのシーケンス番号です。

デフォルトの動作では、1 つのファイルに LOB が書き込まれますが、個々の LOB を別々のファイルに格納するように指定することも可能です。エクスポート・ユーティリティーは、複数の LOB を 1 つのファイルに格納する動作を有効にするために、LOB ロケーション指定子 (LLS) を生成します。この LLS は、LOB データをファイル内のどこに格納するのかを指定したストリングであり、エクスポート出力ファイルに書き込まれます。LLS のフォーマットは、lobfilename.ext.nnn.mmm/ です。ここで、lobfilename.ext は LOB の入ったファイルの名前、nnn はファイル内の LOB のオフセット (バイト単位)、mmm は LOB の長さ (バイト単位) です。例えば、LLS が db2exp.001.123.456/ である場合、これは、LOB がファイル db2exp.001 にあり、ファイル内の 123 バイトのオフセットで始まり、長さが 456 バイトであることを示します。 LLS で示されたサイズが 0 の場合、LOB の長さは 0 であると見なされます。長さが -1 の場合には、LOB は NULL と見なされ、オフセットおよびファイル名は無視されます。

個々の LOB データを同じファイルに連結しない場合は、lobsinsepfiles ファイル・タイプ修飾子を使用して、それぞれの LOB を別々のファイルに書き込むようにします。

注: IXF ファイル・フォーマットには、LOB 列がログ記録されるかどうかなどの、列の LOB オプションが保管されません。そのため、インポート・ユーティリティーでは、1 GB 以上の大きさに定義された LOB 列の入っている表を再作成できません。

例 1
次の例では、LOB を 1 つの DEL ファイルにエクスポートする方法を示しています (エクスポート LOB ファイルのベース名として、lobs1 を指定します)。

```
db2 export to myfile.del of del lobs to mylobs/
   lobfile lobs1 modified by lobsinfile
   select * from emp_photo
```

例 2
次の例では、LOB を 1 つの DEL ファイルにエクスポートし、それぞれの LOB 値を別々のファイルに書き込み、LOB ファイルを 2 つのディレクトリーに格納する方法を示しています。

```
db2 export to myfile.del of del
lobs to /db2exp1/, /db2exp2/ modified by lobsinfile
select * from emp_photo
```

インポート・ユーティリティー

インポートの概要

インポート・ユーティリティーは、SQL INSERT ステートメントを使用して、表、型付き表、ビューにデータを取り込むためのユーティリティーです。インポート・データを受け取る表またはビューにすでにデータが入っている場合は、既存のデータを入力データで置き換えるか、既存のデータに入力データを追加するかのいずれかを選択できます。

エクスポートと同じように、インポートも比較的シンプルなデータ移動ユーティリティーです。このユーティリティーを実行するには、CLP コマンドを実行するか、ADMIN_CMD ストアード・プロシージャーを呼び出すか、ユーザー・アプリケーションで db2Import API を呼び出します。

インポートでサポートされているデータ・フォーマットや、インポートで使用できるフィーチャーがいくつかあります。

- インポートのデータ・フォーマットとしては、IXF、ASC、DEL がサポートされています。
- インポート操作をカスタマイズするために、ファイル・タイプ修飾子を一緒に使用できます。
- インポートを使用して、階層データや型付き表を移動することもできます。
- インポートは、すべてのアクティビティーをログに記録し、索引を更新し、制約を検証し、トリガーを起動します。
- インポートでは、データの挿入先になる表またはビューの列の名前を指定できます。
- インポートは、DB2 Connect でも使用できます。

インポート・モード

インポートには、データのインポート方式を示す 5 つのモードがあります。最初の 3 つ (INSERT、INSERT_UPDATE、REPLACE) は、ターゲット表がすでに存在する場合に使用します。その 3 つとも、データ・フォーマットとして、IXF、ASC、DEL をサポートしています。ただし、ニックネームで使用できるのは、INSERT と INSERT_UPDATEだけです。

表 12. INSERT、INSERT_UPDATE、および REPLACE インポート・モードの概要

モード	最良事例の使用法
INSERT	入力データをターゲット表に挿入します (既存のデータは変更しません)
INSERT_UPDATE	主キー値が一致する行を入力行の値で更新します 一致する行がない場合は、インポート行を表にそのまま挿入します
REPLACE	既存のデータをすべて削除し、インポート・データを挿入します (表と索引の定義はそのまま維持します)

他の 2 つのモード (REPLACE_CREATE、CREATE) は、ターゲット表が存在しない場合に使用します。これらのモードを使用できるのは、入力ファイルが PC/IXF 形式であり、作成する表の構造記述がそのファイルに入っている場合に限られます。オブジェクト表にそれ自体以外の従属表がある場合は、これらのモードでインポートを実行できません。

注: インポートの CREATE モードと REPLACE_CREATE モードは、非推奨になります。代わりに、**db2look** ユーティリティーを使用してください。

表 13. REPLACE_CREATE および CREATE インポート・モードの概要

モード	最良事例の使用法
REPLACE_CREATE	既存のデータをすべて削除し、インポート・データを挿入します (表と索引の定義はそのまま維持します) ターゲット表と索引が存在しない場合は、ターゲット表と索引を作成します
CREATE	ターゲット表と索引を作成します 新しい表を作成する表スペースの名前を指定できます

IBM Data Studio バージョン 3.1 以降では、次のタスクのためにタスク・アシスタントを使用できます: データのインポート. タスク・アシスタントは、オプションの設定、タスク実行のために自動生成されたコマンドの確認、およびそれらのコマンドの実行のプロセスをガイドします。詳しくは、タスク・アシストを使用したデータベースの管理を参照してください。

インポートのしくみ

インポートに必要なステップの数と時間の長さは、移動するデータの量と指定するオプションによって異なります。インポート操作の各ステップは、次のような流れになります。

1. 表をロッキングします。
   既存のターゲット表で並行アクセスを許可するかどうかに応じて、インポートは、そのターゲット表の排他 (X) ロックまたは非排他 (IX) ロックを取得します。
2. データを検出して取り出します
   インポートは、FROM 節を使用して、入力データを検出します。XML データや LOB データが存在することをコマンドで指定していれば、インポートは、その種のデータも検出します。
3. データを挿入します。
   インポートは、既存のデータを置き換えるか、新しいデータ行を表に追加します。
4. 制約をチェックして、トリガーを起動します。
   インポートは、データを書き込むときに、挿入するそれぞれの行が、ターゲット表で定義されている制約に準拠しているかどうかを確認します。リジェクトされた行の情報は、メッセージ・ファイルに書き込まれます。インポートは、既存のトリガーも起動します。

5. 操作をコミットします。
   インポートは、変更内容を保管して、ターゲット表のロックを解放します。インポートの処理中に、コミットを周期的に実行するように指定することも可能です。

基本的なインポート操作で必須の項目は、以下のとおりです。

- 入力ファイルのパスと名前
- ターゲット表またはターゲット・ビューの名前または別名
- 入力ファイル内のデータのフォーマット
- データをインポートする方式
- トラバース順序 (階層データをインポートする場合)
- 副表リスト (型付き表をインポートする場合)

追加のオプション

インポート操作をカスタマイズするために使用できるオプションがいくつかあります。MODIFIED BY 節にファイル・タイプ修飾子を指定すれば、データ・フォーマットを変更したり、インポート・ユーティリティーによるデータ処理の方法を指定したり、パフォーマンスを改善したりすることが可能になります。

インポート・ユーティリティーのデフォルトの動作では、正常なインポートが終了するまでコミットは実行されません (ただし、一部の ALLOW WRITE ACCESS インポートの場合は例外です)。このデフォルトの動作では、インポートの速度は上がりますが、並行性、リスタートのしやすさ、アクティブ・ログのスペースなどの考慮事項からすると、インポートの処理中にコミットを実行するように指定する方が望ましい場合もあります。そのためには、**COMMITCOUNT** パラメーターを「automatic」に設定するという方法があります。この場合、インポートは、コミットを実行するタイミングを内部で決定します。あるいは、**COMMITCOUNT** に特定の数値を設定することも可能です。その場合、インポートは、指定の数のレコードがインポートされるたびにコミットを実行します。

インポートのパフォーマンスを改善するための方法もいくつかあります。インポート・ユーティリティーは、組み込み SQL アプリケーションであり、内部で SQL フェッチを実行するので、SQL 操作に当てはまる最適化の手法は、インポートにも当てはまります。インポートでは、1 行ずつ挿入していくのがデフォルトの動作ですが、compound ファイル・タイプ修飾子を使用すれば、指定の数の行を一挙に挿入することが可能になります。インポートの実行時に多数の警告が生成される (つまり、それだけ操作の速度が低下する) ことが予想される場合は、norowwarnings ファイル・タイプ修飾子を指定して、リジェクトされた行に関する警告を抑制することもできます。

メッセージ・ファイル

インポートの実行時には、その操作に関連したエラー・メッセージ、警告メッセージ、通知メッセージを格納した標準の ASCII テキスト・メッセージ・ファイルが書き出されます。アプリケーション・プログラミング・インターフェース (API) db2Import でユーティリティーを呼び出す場合は、そのファイルの名前を事前に **MESSAGES** パラメーターで指定しておく必要がありますが、それ以外の場合にファイル名を指定するかどうかは任意です。メッ

セージ・ファイルは、インポートの進行状況をモニターするための便利な方法です。メッセージ・ファイルには、インポートの進行中にもアクセスできるからです。インポート操作が失敗した場合は、メッセージ・ファイルを使用して、正常にインポートされた最後の行を確認することによって、リスタート・ポイントを判別できます。

注: リモート・データベースに対するインポート操作で生成される出力メッセージの量が 60 KB を超えると、ユーティリティーは、最初の 30 KB と最後の 30 KB を維持します。

インポートを使用するために必要な特権と権限

ユーザーは、特権によってデータベース・リソースを作成したりアクセスしたりすることが可能になります。権限レベルは、特権をグループ化する手段となるものであり、さらに高水準のデータベース・マネージャーの保守およびユーティリティーのさまざまな操作を提供します。それらの働きにより、データベース・マネージャーとそのデータベース・オブジェクトへのアクセスが制御されます。

ユーザーは、適切な許可 (必要な特権または権限) が付与されているオブジェクトにしかアクセスできません。

DATAACCESS 権限があれば、すべてのタイプのインポート操作が実行できます。インポートのタイプごとに、インポートにかかわるそれぞれの表、ビュー、ニックネームで必要になる他の権限を以下の表にまとめておきます。

表 14. インポート操作を実行するために必要な権限

モード	必要な権限
INSERT	「CONTROL」または 「INSERT および SELECT」
INSERT_UPDATE	「CONTROL」または 「INSERT、SELECT、UPDATE、および DELETE」
REPLACE	「CONTROL」または 「INSERT、SELECT、および DELETE」
REPLACE_CREATE	ターゲット表が存在する場合:「CONTROL」または 「INSERT、SELECT、および DELETE」 ターゲット表が存在しない場合:「CREATETAB (データベース)」、「USE (表スペース)」、および以下 スキーマが存在しない場合:「IMPLICIT_SCHEMA (データベース)」 スキーマが存在する場合:「CREATEIN (スキーマ)」
CREATE	「CREATETAB (データベース)」、「USE (表スペース)」、および以下 スキーマが存在しない場合:「IMPLICIT_SCHEMA (データベース)」 スキーマが存在する場合:「CREATEIN (スキーマ)」

注: **IMPORT** コマンドの **CREATE** オプションと **REPLACE_CREATE** オプションは非推奨で、将来のリリースでは廃止される可能性があります。
さらに、**REPLACE** オプションまたは **REPLACE_CREATE** オプションを使用するには、表をドロップするための権限がセッション許可 ID に必要です。

階層にインポートする場合は、必要な権限がモードによっても変わってきます。階層がすでに存在している場合に **REPLACE** 操作を実行するには、階層内のすべての副表に関する CONTROL 特権が必要です。階層が存在しない場合に **REPLACE_CREATE** 操作を実行するには、階層内のすべての副表に関する CONTROL 特権のほかに CREATETAB と USE が必要です。

さらに、LBAC (ラベル・ベースのアクセス制御) セキュリティー・ラベルが定義されている表にインポートする場合も、いくつかの考慮事項があります。保護列を持つ表にデータをインポートするには、表内のすべての保護列への書き込みアクセスを可能にする LBAC 信用証明情報がセッション許可 ID に必要です。保護行を持つ表にデータをインポートするには、表を保護するセキュリティー・ポリシーの一部である、書き込みアクセス用のセキュリティー・ラベルが、セッション許可 ID に対して付与されていなければなりません。

データのインポート

インポート・ユーティリティーは、サポートされているファイル・フォーマットを用いて、外部ファイルから表、階層、ビュー、またはニックネームにデータを挿入します。

ロード・ユーティリティーは、高速な代替方法ですが、ロード・ユーティリティーでは、階層レベルのデータのロードはサポートされていません。

始める前に

インポート・ユーティリティーを起動するには、その前にデータのインポート先となるデータベースに接続されているか、または暗黙接続が可能な状態になっていなければなりません。暗黙的な接続が可能である場合には、デフォルトのデータベースへの接続が確立されます。

DB2 for Linux、UNIX、または Windows クライアントから DB2 for Linux、UNIX、または Windows データベース・サーバーへのユーティリティーのアクセスは、エンジンからの直接接続でなければなりません。ユーティリティーが DB2 Connect ゲートウェイまたはループバック環境を介してアクセスすることはできません。

このユーティリティーは COMMIT ステートメントまたは ROLLBACK ステートメントを発行するため、インポートの起動前に COMMIT ステートメントまたは ROLLBACK 操作を実行することにより、すべてのトランザクションを完了し、すべてのロックを解除しておいてください。

注: IMPORT コマンドの **CREATE** オプションと **REPLACE_CREATE** パラメーターは非推奨で、将来のリリースでは廃止される可能性があります。

制約事項

インポート・ユーティリティーには、以下の制約事項が適用されます。

- 既存の表が、従属表の外部キーから参照される主キーを備えた親表である場合、そのデータは置換できず、追加だけが可能です。
- REFRESH IMMEDIATE モードで定義されたマテリアライズ照会表の基礎表へのインポート置換操作は実行できません。

- システム表、サマリー表、または構造化タイプ列を持つ表にデータをインポートすることはできません。
- 宣言された一時表にデータをインポートすることはできません。
- インポート・ユーティリティーを介してビューを作成することはできません。
- PC/IXF ファイルから表を作成する場合、参照制約と外部キー定義は保存されません。 (主キー定義は、データが前に SELECT * を使ってエクスポートされた場合、保存されます。)
- インポート・ユーティリティーは独自の SQL ステートメントを生成するので、場合によっては最大ステートメント・サイズの 2 MB を超えることがあります。
- **CREATE** または **REPLACE_CREATE** インポート・パラメーターを使用してパーティション表またはマルチディメンション・クラスタリング表 (MDC) を再作成することはできません。
- XML 列を含む表を再作成することはできません。
- 暗号化されたデータはインポートできません。
- インポート置換操作では、Not Logged Initially 節は考慮されません。**IMPORT** コマンドの **REPLACE** パラメーターは CREATE TABLE ステートメントの NOT LOGGED INITIALLY (NLI) 節や ALTER TABLE ステートメントの ACTIVATE NOT LOGGED INITIALLY 節を考慮しません。 NLI が呼び出される CREATE TABLE ステートメントまたは ALTER TABLE ステートメントと同じトランザクション内で **REPLACE** アクションを伴うインポートを実行する場合、そのインポートでは NLI 節は考慮されません。このシナリオでは、すべての挿入がログに記録されます。

  対処策 1: DELETE ステートメントを使用して表の内容を削除してから、INSERT ステートメントによってインポートを呼び出します。

  対処策 2: 表をドロップしてから再作成した後、INSERT ステートメントによってインポートを呼び出します。

インポート・ユーティリティーに適用される制限: リモート・データベースに対するインポート操作で生成された出力メッセージの量が 60 KB を超える場合、このユーティリティーは最初の 30 KB と最後の 30 KB を維持します。

手順

インポート・ユーティリティーを起動するには、次のようにします。

- コマンド行プロセッサー (CLP) で **IMPORT** コマンドを発行します。
- クライアント・アプリケーションから db2Import アプリケーション・プログラミング・インターフェース (API) を呼び出します。
- **IMPORT** コマンドに関する IBM Data Studio のタスク・アシストを開きます。

例

シンプルなインポート操作の場合に指定する必要があるのは、入力ファイル、ファイル形式、インポート・モード、ターゲット表 (または作成する表の名前) だけです。

例えば、CLP からデータをインポートするには、以下の **IMPORT** コマンドを入力します。

```
db2 import from filename of fileformat import_mode into table
```

filename は、インポートするデータが含まれている入力ファイルの名前、*fileformat* はファイル形式、*import_mode* はモード、*table* は、データの挿入先の表の名前です。

ただし、警告メッセージやエラー・メッセージを書き込むメッセージ・ファイルを指定することも可能です。そのためには、**MESSAGES** パラメーターとメッセージ・ファイル名を追加します。例えば、以下のようにします。

```
db2 import from filename of fileformat messages messagefile import_mode into table
```

XML データのインポート

インポート・ユーティリティーを使用することにより、DB2 for Linux, UNIX, and Windows ソース・データ・オブジェクトの表名またはニックネームを使用して XML データを XML 表列にインポートできます。

データを XML 表列にインポートするとき、XML FROM オプションを使用して入力 XML データ (1 つまたは複数) のパスを指定できます。例えば、以前にエクスポートされた XML ファイル "/home/user/xmlpath/xmldocs.001.xml" では、以下のコマンドを使用してデータを表にインポートして戻すことができます。

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath INSERT INTO USER.T1
```

スキーマに対して挿入された文書を検証する

XMLVALIDATE オプションにより、XML 文書がインポートされる際に、それらが XML スキーマに準拠しているかどうかを検証できます。次の例では、XML 文書がエクスポートされた時点で保存されたスキーマ情報に準拠しているかどうかに関して、着信 XML 文書が検証されます。

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath XMLVALIDATE
     USING XDS INSERT INTO USER.T1
```

構文解析オプションの指定

XMLPARSE オプションを使用して、インポートされた XML 文書の空白文字を保存するかストリップするかを指定できます。次の例では、XML 文書がエクスポートされたときに保管された XML スキーマ情報に対してすべてのインポートされた XML 文書が検証されて、これらの文書は空白文字を保存しながら構文解析されます。

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath XMLPARSE PRESERVE
WHITESPACE XMLVALIDATE USING XDS INSERT INTO USER.T1
```

インポート・セッション - CLP の例

例 1

次に示すのは myfile.ixf から STAFF 表に情報をインポートする方法の例です。

```
db2 import from myfile.ixf of ixf messages msg.txt insert into staff

SQL3150N  The H record in the PC/IXF file has product "DB2    01.00", date
"19970220", and time "140848".
```

```
SQL3153N  The T record in the PC/IXF file has name "myfile",
qualifier "        ", and source "            ".

SQL3109N  The utility is beginning to load data from file "myfile".

SQL3110N  The utility has completed processing.  "58" rows were read from the
input file.

SQL3221W  ...Begin COMMIT WORK. Input Record Count = "58".

SQL3222W  ...COMMIT of any database changes was successful.

SQL3149N  "58" rows were processed from the input file.  "58" rows were
successfully inserted into the table.  "0" rows were rejected.
```

例 2

次の例では、ID 列が含まれている表にインポートする方法を示します。

TABLE1 には、以下の 4 つの列があります。

- C1 VARCHAR(30)
- C2 INT GENERATED BY DEFAULT AS IDENTITY
- C3 DECIMAL(7,2)
- C4 CHAR(1)

TABLE2 は、C2 が GENERATED ALWAYS ID 列であることを除き、 TABLE1 と同じです。

DATAFILE1 内のデータ・レコード (DEL フォーマット):

```
"Liszt"
"Hummel",,187.43, H
"Grieg",100, 66.34, G
"Satie",101, 818.23, I
```

DATAFILE2 内のデータ・レコード (DEL フォーマット):

```
"Liszt", 74.49, A
"Hummel", 0.01, H
"Grieg", 66.34, G
"Satie", 818.23, I
```

以下のコマンドでは、行 1 および 2 の ID 値が生成されます。これは、DATAFILE1 内にこれらの行の ID 値が存在しないためです。ただし、行 3 および 4 には、ユーザー提供の ID 値である 100 および 101 がそれぞれ割り当てられます。

```
db2 import from datafile1.del of del replace into table1
```

DATAFILE1 を TABLE1 にインポートして、すべての行の ID 値が生成されるようにするには、以下のコマンドのいずれかを発行してください。

```
db2 import from datafile1.del of del method P(1, 3, 4)
   replace into table1 (c1, c3, c4)
db2 import from datafile1.del of del modified by identityignore
   replace into table1
```

DATAFILE2 を TABLE1 にインポートして、それぞれの行ごとに ID 値が生成されるようにするには、以下のコマンドのいずれかを発行してください。

```
db2 import from datafile2.del of del replace into table1 (c1, c3, c4)
db2 import from datafile2.del of del modified by identitymissing
   replace into table1
```

識別に関係するファイル・タイプ修飾子を使用せずに DATAFILE1 を TABLE2 にインポートすると、行 1 と 2 は挿入されますが、行 3 と 4 はリジェクトされます。これは、行 3 と 4 では独自に非 NULL 値が提供されており、 ID 列が GENERATED ALWAYS であるためです。

例 3

次の例では、NULL 標識が含まれている表にインポートする方法を示します。

TABLE1 には以下に示す 5 つの列があります。

- COL1 VARCHAR 20 NOT NULL WITH DEFAULT
- COL2 SMALLINT
- COL3 CHAR 4
- COL4 CHAR 2 NOT NULL WITH DEFAULT
- COL5 CHAR 2 NOT NULL

ASCFILE1 には以下に示す 6 つのエレメントがあります。

- ELE1、位置 01 から 20
- ELE2、位置 21 から 22
- ELE5、位置 23 から 23
- ELE3、位置 24 から 27
- ELE4、位置 28 から 31
- ELE6、位置 32 から 32
- ELE6、位置 33 から 40

データ・レコード:

```
1...5....10...15...20...25...30...35...40
Test data 1         XXN 123abcdN
Test data 2 and 3   QQY    wxyzN
Test data 4,5 and 6 WWN6789    Y
```

以下のコマンドは、ASCFILE1 から TABLE1 にレコードをインポートします。

```
db2 import from ascfile1 of asc
method L (1 20, 21 22, 24 27, 28 31)
null indicators (0, 0, 23, 32)
insert into table1 (col1, col5, col2, col3)
```

注:

1. COL4 は入力ファイルにはないので、そのデフォルト値 (NOT NULL WITH DEFAULT と定義されている) を使用して TABLE1 に挿入されます。
2. 位置 23 および 32 は、TABLE1 の COL2 と COL3 に NULL をロードするかどうかを行ごとに示すために使用されます。特定のレコードのうち列の NULL 標識位置が `Y` なら、その列は NULL になります。それが `N` の場合は、入力レコードのその列のデータ位置 (L(.....) で定義) にあるデータ値が、その行の列データのソースとして使用されます。この例では、行 1 のどの列も NULL ではなく、行 2 の COL2 は NULL であり、行 3 の COL3 は NULL です。

3. この例では、COL1 と COL5 の NULL INDICATORS は 0 (ゼロ) として指定されますが、それはそのデータを NULL 不可能であることを示しています。
4. 特定の列に対する NULL INDICATOR は入力レコードのどの位置でも可能ですが、その位置は必ず指定しなければならず、 Y または N のいずれかの値が提供される必要があります。

インポート表の再作成

インポート・ユーティリティーの CREATE モードを使用すると、エクスポート・ユーティリティーによって保存された表を再作成することができます。ただし、この処理にはいくつかの制約があり、入力表の属性の多くが保持されません。

インポート時に表を再作成するには、エクスポート操作でいくつかの要件を満たしておく必要があります。まず、元の表を IXF ファイルにエクスポートしなければなりません。ファイル形式 DEL または ASC のファイルをエクスポートした場合は、出力ファイルにレコード・データは組み込まれますが、ターゲット表の記述は組み込まれません。このようなファイル・フォーマットのデータを持つ表を再作成するには、ターゲット表を作成してから、ロードまたはインポート・ユーティリティーを使用して、これらのファイルから表にデータを入れます。元の表定義をキャプチャーし、対応するデータ定義言語 (DDL) を生成するには、**db2look** ユーティリティーを使用します。さらに、エクスポート時に使用する SELECT ステートメントには、特定のアクション・ストリングしか組み込めません。例えば、SELECT 節では列名を使用できません。使用できるのは、SELECT * だけです。

注: インポートの CREATE モードは、非推奨になります。表のキャプチャーと再作成には、**db2look** ユーティリティーを使用してください。

保持される属性

元の表の属性のうち、表の再作成時に保持される属性は、以下のとおりです。

- 主キーの名前および定義
- 以下のような列情報
  - 列名
  - 列データ・タイプ。ユーザー定義の特殊タイプはその基本タイプとして保存されます。
  - ID プロパティー
  - 長さ (lob_file タイプの場合を除く)
  - コード・ページ (該当する場合)
  - ID オプション
  - 列が NULL 可能または不可のどちらで定義されているか
  - 定数のデフォルト値 (ある場合)。ただし他のタイプのデフォルト値は該当しません。
- 以下のような索引情報
  - 索引名
  - 索引の作成者名
  - 列名と、昇順または降順のどちらで各列がソートされるか

- 索引がユニーク索引として定義されているかどうか
- 索引がクラスター化されているかどうか
- 索引で反転スキャンが可能かどうか
- PCTFREE 値
- MINPCTUSED 値

注: 索引の列名に文字 - または + が含まれていると、索引情報は保持されません。その場合は、SQL27984W が返されます。

失われる属性

元の表の属性のうち、表の再作成時に保持されない属性は、以下のとおりです。

- ソースが、通常の表、マテリアライズ照会表 (MQT)、ビュー、またはこれらのソースの全部または一部から取られた列セットのうちのどれであったか
- ユニーク制約およびその他のタイプの制約、またはトリガー (主キー制約を含まない)
- 以下のような表情報
  - MQT 定義 (該当する場合)
  - MQT オプション (該当する場合)
  - 表スペース・オプション。ただしこの情報は、**IMPORT** コマンドを使って指定することができます。
  - マルチディメンション・クラスタリング (MDC) ディメンション
  - パーティション表ディメンション
  - 表パーティション・キー
  - NOT LOGGED INITIALLY プロパティー
  - チェック制約
  - 表のコード・ページ
  - 保護表のプロパティー
  - 表または値の圧縮オプション
- 以下のような列情報
  - 定数値以外の任意のデフォルト値
  - LOB オプション (ある場合)
  - XML プロパティー
  - CREATE TABLE ステートメントの references 節 (ある場合)
  - 参照制約 (ある場合)
  - チェック制約 (ある場合)
  - 生成列オプション (ある場合)
  - データベースの有効範囲シーケンスに依存する列
  - 暗黙的な非表示プロパティー
- 以下のような索引情報
  - INCLUDE 列 (存在する場合)

- 索引が主キー索引である場合は、索引名
- 索引が主キー索引である場合は、降順のキー (デフォルトは昇順)
- 索引列名に 16 進値 0x2B または 0x2D が含まれる
- コード・ページ変換後に 128 バイトを超える索引名
- PCTFREE2 値
- ユニーク制約

注: このリストですべてを取り上げているわけではないので、このリストを使用するときには注意が必要です。

インポートが失敗し、SQL3311N が返された場合でも、ファイル・タイプ修飾子 `forcecreate` を使用すれば、表を再作成できます。この修飾子を使用すると、情報が欠落したまま、限られた情報だけで表が作成されます。

型付き表のインポートに関する考慮事項

インポート・ユーティリティーを使用すると、既存のデータ階層を保持しながら、型付き表から (および型付き表の中へ) データを移動できます。必要に応じて、インポート時に表階層と型階層を作成することも可能です。

ある階層構造の型付き表から別の階層構造の型付き表にデータを移動するには、エクスポート操作で特定のトラバース順序を指定し、中間のフラット・ファイルを作成します。一方、インポート・ユーティリティーでは、**CREATE**、**INTO** *table-name*、**UNDER**、および **AS ROOT TABLE** の各パラメーターによって、移動する階層のサイズと配置を制御します。さらに、インポートでは、ターゲット・データベースに格納する内容も制御します。例えば、それぞれの副表名の最後に属性リストを指定することによって、ターゲット・データベースに移動する属性を制限できます。属性リストを使用しない場合は、それぞれの副表のすべての列が移動されます。

表の再作成

実行できるインポートのタイプは、入力ファイルのファイル形式によって異なります。ASC または DEL のデータを操作する場合は、データをインポートする前にターゲット表または階層が存在している必要があります。一方、PC/IXF ファイルのデータは、表または階層が存在していない場合でも、**CREATE** のインポート操作を指定することによってインポートできます。ただし、**CREATE** オプションを指定しても、インポート時に副表の定義を変更することはできません。

トラバース順序

入力ファイルに含まれているトラバース順序によって、データ階層を維持することが可能になります。したがって、エクスポート・ユーティリティーを起動する場合とインポート・ユーティリティーを起動する場合では、同じトラバース順序を使用しなければなりません。

PC/IXF ファイル形式の場合は、ターゲットの副表の名前を指定する必要があるだけで、トラバース順序については、ファイルに格納されているデフォルトのトラバース順序が使用されます。

型付き表で **CREATE** 以外のオプションを使用する場合は、トラバース順序リストによってトラバース順序を指定できます。このユーザー指定のトラバース順序は、エクスポート操作時に使用されたトラバース順序と一致していな

ければなりません。インポート・ユーティリティーは、以下の条件が満たされていれば、ターゲット・データベースにデータを正確に移動できます。

- ソースとターゲットの両方のデータベースで、副表の定義が同じである。
- ソースとターゲットの両方のデータベースで、副表間の階層リレーションシップが同じである。
- トラバース順序が同じである。

トラバース順序を定義する場合、開始点と階層を下るパスを決定できますが、それぞれの分岐の最後までトラバースが終わってからでないと、階層内の次の分岐のトラバースを開始できません。インポート・ユーティリティーは、指定されたトラバース順序がこの条件に違反していないかどうかを検査します。

例

この項の例は、以下の 4 つの有効なトラバース順序がある階層構造に基づいています。

- Person、Employee、Manager、Architect、Student
- Person、Student、Employee、Manager、Architect
- Person、Employee、Architect、Manager、Student
- Person、Student、Employee、Architect、Manager

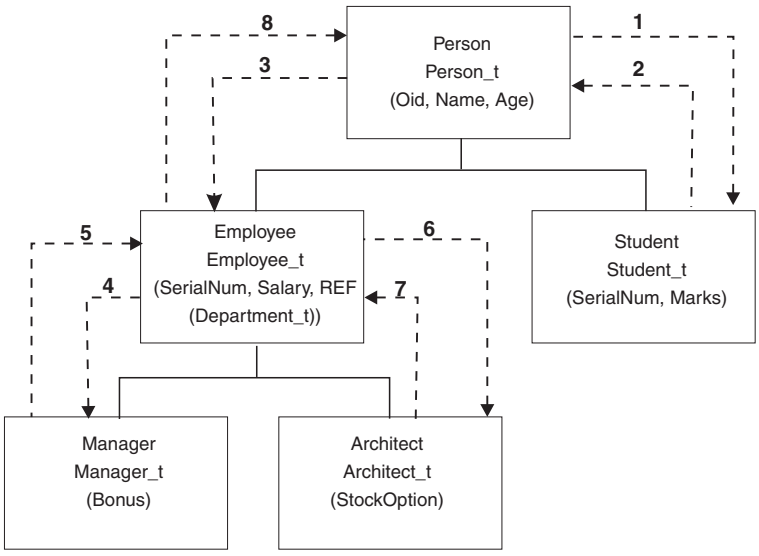


図 2. 階層の例

例 1

インポートを使用して階層 (前のエクスポート操作で作成したデータ・ファイル `entire_hierarchy.ixf` に含まれている階層) 全体を再作成するには、以下のコマンドを入力します。

```
DB2 CONNECT TO Target_db
DB2 IMPORT FROM entire_hierarchy.ixf OF IXF CREATE INTO
   HIERARCHY STARTING Person AS ROOT TABLE
```

階層内のそれぞれの型が存在しない場合は、それが作成されます。型がすでに存在している場合は、ターゲット・データベースとソース・データベースとで同じ定義でなければなりません。もし同じでなければ、SQL エラー (SQL20013N) が戻されます。ここでは新たに階層を作成しているので、ターゲット・データベース (Target_db) に移動されるデータ・ファイルで定義されている副表がその中に存在していてはなりません。ソース・データベース階層のすべての表が新たに作成されます。ソース・データベースから移されるデータは、ターゲット・データベース内の適切な副表にインポートされます。

例 2
ソース・データベースの階層全体を再作成し、その階層をターゲット・データベースにインポートするときに、選択したデータだけを維持するには、以下のコマンドを入力します。

```
DB2 CONNECT TO Target_db
DB2 IMPORT FROM entire_hierarchy.del OF DEL INSERT INTO (Person,
   Employee(Salary), Architect) IN HIERARCHY (Person, Employee,
   Manager, Architect, Student)
```

ターゲット表 PERSON、EMPLOYEE、および ARCHITECT はすでに存在していなければなりません。データは副表 PERSON、EMPLOYEE、および ARCHITECT の中にインポートされます。つまり、次のようにインポートされます。

- PERSON 内のすべての列を PERSON へ
- PERSON のすべての列、および EMPLOYEE 内の SALARY を EMPLOYEE へ
- PERSON のすべての列、EMPLOYEE 内の SALARY、および ARCHITECT のすべての列を ARCHITECT へ

SerialNum 列と REF(Employee_t) 列は、EMPLOYEE またはその副表 (つまり、データがインポートされている唯一の副表である ARCHITECT) にインポートされません。

注: ARCHITECT は EMPLOYEE の副表であり、EMPLOYEE に指定されている唯一のインポート列は SALARY なので、ARCHITECT にインポートされる EMPLOYEE 固有の列は SALARY だけになります。つまり、SerialNum 列と REF(Employee_t) 列はいずれも、EMPLOYEE 行や ARCHITECT 行にインポートされません。
表 MANAGER と表 STUDENT のデータはインポートされません。

例 3
次の例では、正規の表からエクスポートした後、ある階層内の 1 つの副表としてインポートします。 **EXPORT** コマンドが正規の表 (非型付き表) で実行されるので、データ・ファイルの中に Type_id 列はありません。これはファイル・タイプ修飾子 no_type_id を指定して示します。そうすれば、インポート・ユーティリティーは、最初の列が Type_id 列であることを予期しなくなります。

```
DB2 CONNECT TO Source_db
DB2 EXPORT TO Student_sub_table.del OF DEL SELECT * FROM
   Regular_Student
DB2 CONNECT TO Target_db
DB2 IMPORT FROM Student_sub_table.del OF DEL METHOD P(1,2,3,5,4)
   MODIFIED BY NO_TYPE_ID INSERT INTO HIERARCHY (Student)
```

この例では、ターゲット表 STUDENT がすでに存在していなければなりません。STUDENT は副表なので、最初の列に Type_id が存在しないことを示すために、修飾子 no_type_id を使用します。しかし、Object_id 列、および STUDENT 表内の他のすべての属性が存在することを確認する必要があります。 STUDENT 表にインポートされるどの行でも、その最初の列は Object-id であると見なされます。**METHOD** 節によって、最後の 2 つの属性の順序が逆転します。

LBAC で保護されたデータのインポートに関する考慮事項

保護行が含まれている表へのインポート操作を正常に実行するには、LBAC (ラベル・ベースのアクセス制御) 信用証明情報が必要です。さらに、ターゲット表に関連付けられているセキュリティー・ポリシーで有効なセキュリティー・ラベル、または有効なセキュリティー・ラベルに変換できるセキュリティー・ラベルを用意することも必要です。

有効な LBAC 信用証明情報がなければ、インポートは失敗し、エラー (SQLSTATE 42512) が返されます。入力データにセキュリティー・ラベルが含まれていない場合や、セキュリティー・ラベルが内部のバイナリー・フォーマットでない場合は、いくつかのファイル・タイプ修飾子を使用して、インポート操作を進めることができます。

保護行が含まれている表にデータをインポートする場合は、ターゲット表にデータ・タイプ DB2SECURITYLABEL の列が 1 つ含まれています。入力データ行にその列の値が含まれていなければ、その行はリジェクトされます。ただし、インポート・コマンドに usedefaults ファイル・タイプ修飾子が指定されている場合は例外です。その場合に、その表を保護しているセキュリティー・ポリシーで書き込みアクセスが認められているセキュリティー・ラベルがあれば、そのセキュリティー・ラベルが使用されます。書き込みアクセスが認められているセキュリティー・ラベルがなければ、その行はリジェクトされ、処理が次の行に移ります。

保護行が含まれている表にデータをインポートする場合に、入力データにデータ・タイプ DB2SECURITYLABEL の列の値が含まれていれば、その表にデータを挿入する場合と同じ規則が当てはまります。インポート対象の行を保護しているセキュリティー・ラベル (データ・ファイル内のその行にあるセキュリティー・ラベル) が書き込み可能であれば、その行の保護のためにそのセキュリティー・ラベルが使用されます。(つまり、そのセキュリティー・ラベルがデータ・タイプ DB2SECURITYLABEL の列に書き込まれます。)そのセキュリティー・ラベルで保護されている行への書き込みができない場合の動作は、ソース表を保護しているセキュリティー・ポリシーの作成方法によって異なります。

- ポリシーを作成した CREATE SECURITY POLICY ステートメントにオプション RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL が含まれていた場合には、挿入は失敗し、エラーが戻されます。
- CREATE SECURITY POLICY ステートメントにそのオプションが組み込まれていなかった場合や、 OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL オプションが代わりに組み込まれていた場合は、データ・ファイルに含まれているその行のためのセキュリティー・ラベルは無視されます。書き込みアクセスが認められているセキュリティー・ラベルがあれば、その行の保護のためにそのセキュリテ

ィー・ラベルが使用されます。この場合、エラーや警告は出されません。書き込みアクセスが認められているセキュリティー・ラベルがなければ、その行はリジェクトされ、処理が次の行に移ります。

区切り文字に関する考慮事項

データ・タイプ DB2SECURITYLABEL の列にデータをインポートする場合、デフォルトでは、データ・ファイル内の値は、そのセキュリティー・ラベルの内部表記を構成する実際のバイトと見なされます。ただし、生データには改行文字が含まれている場合があり、その場合、IMPORT コマンドは、その改行文字を行の区切り文字と誤解してしまう可能性があります。この問題がある場合は、行区切り文字よりもストリング区切り文字を優先するために、`delprioritychar` ファイル・タイプ修飾子を使用します。`delprioritychar` を使用すると、ストリング区切り文字の中にあるレコード区切り文字または列区切り文字は、区切り文字と見なされなくなります。`delprioritychar` ファイル・タイプ修飾子は、改行文字が含まれている値が存在しない場合に使用しても差し支えありません。ただし、インポートの速度は少し低下します。

インポート対象のデータが ASC 形式の場合は、インポートされたセキュリティー・ラベルやセキュリティー・ラベル名に末尾空白が組み込まれる事態を回避するために、追加の手順を実行できます。ASCII フォーマットでは、列位置が区切りとして使用されるので、可変長フィールドにインポートするときにこの問題が発生する可能性があります。末尾ブランク・スペースを切り捨てるには、`striptblanks` ファイル・タイプ修飾子を使用します。

非標準のセキュリティー・ラベル値

セキュリティー・ラベルの値が、セキュリティー・ラベルの各コンポーネントの値を含んだストリングになっている場合、例えば、*S:(ALPHA,BETA)* のようになっている場合でも、データ・ファイルをインポートできます。そのためには、ファイル・タイプ修飾子 `seclabelchar` を使用する必要があります。 `seclabelchar` を使用すると、データ・タイプ DB2SECURITYLABEL の列の値は、セキュリティー・ラベル用のストリング・フォーマットで記述されたセキュリティー・ラベルを含む、ストリング定数と見なされます。ストリングが正しい形式でなければ、その行は挿入されず、警告 (SQLSTATE 01H53) が返されます。そのストリングが、表を保護しているセキュリティー・ポリシーに組み込まれている有効なセキュリティー・ラベルを記述したものでない場合も、その行は挿入されず、警告 (SQLSTATE 01H53) が返されます。

セキュリティー・ラベル列の値がセキュリティー・ラベル名になっている場合も、データ・ファイルをインポートできます。この種のファイルをインポートするには、ファイル・タイプ修飾子 `seclabelname` を使用する必要があります。`seclabelname` を使用すると、データ・タイプ DB2SECURITYLABEL の列のすべての値は、既存のセキュリティー・ラベルの名前を含んだストリング定数と見なされます。表を保護しているセキュリティー・ポリシーでその名前のセキュリティー・ラベルが存在しなければ、その行は挿入されず、警告 (SQLSTATE 01H53) が返されます。

例

すべての例で、入力データ・ファイル myfile.del は DEL 形式になっています。すべての場合のデータのインポート先は、以下のステートメントで作成された REPS という名前の表です。

```
create table reps (row_label db2securitylabel,
id integer,
name char(30))
security policy data_access_policy
```

次の例の入力ファイルには、デフォルト形式のセキュリティー・ラベルが含まれていると見なされます。

```
db2 import from myfile.del of del modified by delprioritychar insert into reps
```

次の例の入力ファイルには、セキュリティー・ラベル・ストリング・フォーマットのセキュリティー・ラベルが含まれていると見なされます。

```
db2 import from myfile.del of del modified by seclabelchar insert into reps
```

次の例の入力ファイルには、セキュリティー・ラベル列のセキュリティー・ラベル名が含まれていると見なされます。

```
db2 import from myfile.del of del modified by seclabelname insert into reps
```

バッファー挿入インポート

パーティション・データベース環境では、インポート・ユーティリティーでバッファー挿入を使用可能にすることができます。そうすることにより、データのインポート時に発生するメッセージングが少なくなり、パフォーマンスが向上します。

バッファー挿入オプションを有効にすると、失敗したバッファー挿入に関する詳細が返されなくなるので、そのオプションを有効にするのは、エラー・レポートに関する心配がない場合に限ってください。

バッファー挿入を使用すると、インポートではデフォルトの **WARNINGCOUNT** 値が 1 に設定されます。そのため、行が 1 つでもリジェクトされると、操作は失敗します。レコードがリジェクトされた場合、ユーティリティーは現在のトランザクションをロールバックします。コミット済みのレコード数を調べれば、どのレコードが正常にデータベースに挿入されたかを判別することができます。コミット済みのレコード数がゼロ以外になりうるのは、COMMITCOUNT オプションを指定していた場合のみです。

別の **WARNINGCOUNT** 値を IMPORT コマンドに明示的に指定していた場合にいずれかの行がリジェクトされると、ユーティリティーからの行サマリー出力は誤っていることがあります。その原因は、バッファー挿入で使用される非同期エラー・レポートと、行グループの挿入時にエラーが検出されたことに起因したそのグループのすべての取り消し行が組み合わさったことにあります。どの入力レコードがリジェクトされたかに関するユーティリティーからのレポートは信頼できないので、どのレコードがコミット済みで、どのレコードをデータベースに再挿入する必要があるかを判別するのは困難になります。

バッファー挿入を要求するには、DB2 バインド・ユーティリティーを使用します。INSERT BUF オプションを使用して、データベースに対してインポート・パッケージ db2uimpm.bnd を再バインドする必要があります。例えば、以下のようにします。

```
db2 connect to your_database
db2 bind db2uimpm.bnd insert buf
```

バッファー挿入フィーチャーと INSERT_UPDATE モードのインポート操作を併用することはできません。バインド・ファイル db2uImpInsUpdate.bnd は、この制限を課します。 INSERT BUF オプションを指定してこのファイルをバインドするべきではありません。そのようにすると、INSERT_UPDATE モードのインポート操作は失敗します。ただし、INSERT、REPLACE、REPLACE_CREATE のいずれかのモードのインポート操作は、この新しいファイルのバインドによる影響を受けません。

ID 列のインポートに関する考慮事項

インポート・ユーティリティーでは、入力データに ID 列の値があるかどうかにかかわりなく、ID 列が含まれている表にデータをインポートできます。

ID 関連のファイル・タイプ修飾子が使用されない場合、このユーティリティーは次のような規則に従って動作します。

- ID 列が GENERATED ALWAYS の場合、入力ファイル内の対応する行の中に ID 列用の値がないか、または NULL 値が明示的に指定されているときに、表の行用の ID 値が生成されます。 ID 列に非 NULL 値を指定すると、その行はリジェクトされます (SQL3550W)。
- ID 列が GENERATED BY DEFAULT の場合、ユーザー提供値が指定されていれば、インポート・ユーティリティーはその値を使用します。データが欠落しているかまたは明示的に NULL であれば、値が生成されます。

インポート・ユーティリティーは、ユーザー提供の ID 値に関して、 ID 列のデータ・タイプ (SMALLINT、INT、BIGINT、または DECIMAL) の値に対して通常行う以外の余分な妥当性検査を行いません。値が重複していても報告されません。しかも、 ID 列を持つ表へのデータのインポート時には compound=*x* 修飾子を使用できません。

ID 列が含まれている表にデータをインポートする操作を簡略化するための 2 つの方法があります。つまり、ファイル・タイプ修飾子として、identitymissing と identityignore を使用できます。

ID 列のないデータのインポート

identitymissing 修飾子は、入力データ・ファイルに ID 列の値が入っていない (NULL すらない) 場合に、 ID 列を持つ表のインポートを容易にします。例えば、次のような SQL ステートメントで定義された表があるとします。

```
create table table1 (c1 char(30),
                     c2 int generated by default as identity,
                     c3 real,
                     c4 char(1))
```

ユーザーが、データをファイル (import.del) から TABLE1 にインポートするとします。このとき、このデータは、 ID 列を持たない表からエクスポートされたものであると想定します。以下に、このようなファイルの例を示します。

```
Robert, 45.2, J
Mike, 76.9, K
Leo, 23.4, I
```

このファイルをインポートする 1 つの方法は、次のように **IMPORT** コマンドを使用して、インポートする列を明示的にリストすることです。

```
db2 import from import.del of del replace into table1 (c1, c3, c4)
```

ただし、多くの列を持つ表の場合、この構文は扱いにくく、誤りを生じる可能性があります。ファイルをインポートする別の方法は、次のように identitymissing ファイル・タイプ修飾子を使うことです。

```
db2 import from import.del of del modified by identitymissing
   replace into table1
```

ID 列のあるデータのインポート

identityignore 修飾子は、ある意味では identitymissing 修飾子の反対の役割を持ちます。この修飾子を指定すると、インポート・ユーティリティーは、入力データ・ファイルに ID 列用の値が入っていても、そのデータを無視して、各行ごとに ID 値を生成します。例えば、上記で定義したように、ユーザーが次のようなデータをファイル (import.del) から TABLE1 にインポートするとします。

```
Robert, 1, 45.2, J
Mike, 2, 76.9, K
Leo, 3, 23.4, I
```

ユーザー指定値の 1、2、および 3 が ID 列に使われない場合、ユーザーは次のような **IMPORT** コマンドを使うことができます。

```
db2 import from import.del of del method P(1, 3, 4)
   replace into table1 (c1, c3, c4)
```

ここでも、表に多くの列があると、このアプローチは扱いにくく、誤りを生じる可能性があります。次のように identityignore 修飾子を使用すると、構文が単純化されます。

```
db2 import from import.del of del modified by identityignore
   replace into table1
```

ID 列を持つ表を IXF ファイルにエクスポートするときには、 **IMPORT** コマンドの REPLACE_CREATE および CREATE オプションを使用して、 ID 列プロパティーを備えた表を再作成することができます。タイプ GENERATED ALWAYS の ID 列の入った表からこのような IXF ファイルが作成されている場合、 identityignore 修飾子を指定するのが、データ・ファイルのインポートを正常に完了する唯一の方法です。このようにしなければ、すべての行がリジェクトされます (SQL3550W)。

注: IMPORT コマンドの CREATE オプションと REPLACE_CREATE オプションは非推奨で、将来のリリースでは廃止される可能性があります。

生成列のインポートに関する考慮事項

インポート・ユーティリティーでは、入力データに生成列の値があるかどうかにかかわりなく、ID 列以外の生成列が含まれている表にデータをインポートできます。

生成列関連のファイル・タイプ修飾子が使用されない場合、インポート・ユーティリティーは次のような規則に従って動作します。

- 生成列の値が生成されるのは、入力ファイル内の対応する行にその列の値が存在しない場合か、NULL 値が明示的に指定されている場合です。生成列に非 NULL 値を指定すると、その行はリジェクトされます (SQL3550W)。

- サーバーが NULL 可能でない生成列に NULL 値を生成すると、そのフィールドが属するデータ行はリジェクトされます (SQL0407N)。これが起きるのは、例えば、NULL 可能列でない生成列が 2 つの表の列の合計として定義されていて、それらの表の列に入力ファイル内で NULL 値が指定された場合です。

生成列が含まれている表にデータをインポートする操作を簡略化するための 2 つの方法があります。つまり、ファイル・タイプ修飾子として、generatedmissing と generatedignore を使用できます。

生成列のないデータのインポート

generatedmissing 修飾子は、表内に存在する生成列の値が入力データ・ファイル内に入っていない (NULL すらない) 場合に、生成列を持つ表へのデータ・インポートを容易にします。例えば、次のような SQL ステートメントで定義された表があるとします。

```
create table table1 (c1 int,
                     c2 int,
                     g1 int generated always as (c1 + c2),
                     g2 int generated always as (2 * c1),
                     c3 char(1))
```

ユーザーが、データをファイル (load.del) から TABLE1 にインポートするとします。このとき、このデータは、生成列を持たない表からエクスポートされたものであると想定します。以下に、このようなファイルの例を示します。

```
1, 5, J
2, 6, K
3, 7, I
```

このファイルをインポートする 1 つの方法は、次のように **IMPORT** コマンドを使用して、インポートする列を明示的にリストすることです。

```
db2 import from import.del of del replace into table1 (c1, c2, c3)
```

ただし、多くの列を持つ表の場合、この構文は扱いにくく、誤りを生じる可能性があります。このファイルをインポートするための別の方法は、generatedmissing ファイル・タイプ修飾子を以下のようにして使用することです。

```
db2 import from import.del of del modified by generatedmissing
   replace into table1
```

生成列のあるデータのインポート

generatedignore 修飾子は、ある意味では generatedmissing 修飾子の反対の役割を持ちます。この修飾子を指定すると、インポート・ユーティリティーは、すべての生成列用のデータが入力データ・ファイルに入っていても、そのデータを無視して、各行ごとに値を生成します。例えば、上記で定義したように、ユーザーが次のようなデータをファイル (import.del) から TABLE1 にインポートするとします。

```
1, 5, 10, 15, J
2, 6, 11, 16, K
3, 7, 12, 17, I
```

ユーザー提供の非 NULL 値 10、11、12 (g1 の場合)、および 15、16、17 (g2 の場合) により、行はリジェクトされます (SQL3550W)。リジェクトされないようにするには、次のような **IMPORT** コマンドを発行します。

```
db2 import from import.del of del method P(1, 2, 5)
   replace into table1 (c1, c2, c3)
```

ここでも、表に多くの列があると、このアプローチは扱いにくく、誤りを生じる可能性があります。次のように generatedignore 修飾子を使用すると、構文が単純化されます。

```
db2 import from import.del of del modified by generatedignore
   replace into table1
```

INSERT_UPDATE の場合、生成列が主キー列でもあり、generatedignore 修飾子が指定されていれば、**IMPORT** コマンドは、generatedignore 修飾子を有効と見なします。 **IMPORT** コマンドが UPDATE ステートメントの WHERE 節でこの列をユーザー提供値に置換することはありません。

LOB のインポートに関する考慮事項

インポート・ユーティリティーでは、1 つの列値のサイズが 32 KB までに制限されているので、LOB をインポートするときには、追加の考慮事項について検討する必要があります。

デフォルトの動作では、インポート・ユーティリティーは、入力ファイル内のデータを、列にロードするデータとして処理します。ただし、メインの入力データ・ファイルにラージ・オブジェクト (LOB) データが格納されていると、データのサイズが 32 KB に制限されます。したがって、データを失わないようにするには、LOB データをメインのデータ・ファイルとは別の場所に格納し、LOB をインポートするときに lobsinfile ファイル・タイプ修飾子を指定する必要があります。

LOBS FROM 節を指定すると、lobsinfile が暗黙的にアクティブ化されます。LOBS FROM 節によって、データのインポート時に LOB ファイルを検索するパスのリストをインポート・ユーティリティーに渡します。LOBS FROM オプションが指定されない場合、インポートする LOB ファイルは、入力リレーショナル・データ・ファイルと同じパスにあると見なされます。

LOB データの格納場所の指定

LOB ロケーション指定子 (LLS) を使用すれば、LOB 情報のインポート時に複数の LOB を 1 つのファイルに格納できます。lobsinfile を指定すると、エクスポート・ユーティリティーは、LOB データの格納場所を示す指定子を生成して、エクスポート出力ファイルに格納します。MODIFIED BY lobsinfile オプションを指定したデータをインポートする場合、データベースは、対応する LOB 列ごとに LLS が存在することを予期します。 LLS 以外のものが LOB 列にある場合には、データベースはこれを LOB ファイルと見なし、ファイル全体を LOB としてロードします。

CREATE モードのインポートでは、LONG IN 節を使用して、LOB データを作成して別の表スペースに格納するように指定することも可能です。

次の例では、LOB が別のファイルに格納されている状態で DEL ファイルをインポートする方法を示します。

```
IMPORT FROM inputfile.del OF DEL
 LOBS FROM /tmp/data
 MODIFIED BY lobsinfile
 INSERT INTO newtable
```

ユーザー定義特殊タイプのインポートに関する考慮事項

インポート・ユーティリティーは、ユーザー定義特殊タイプ (UDT) をそれと類似の基本データ・タイプに自動的にキャストします。それにより、UDT を基本データ・タイプに明示的にキャストする手間が省けます。キャストによって、SQL で UDT と基本データ・タイプとの間の比較が可能になります。

インポートに関する追加の考慮事項

クライアント/サーバー環境とインポート

リモート・データベースにファイルをインポートする場合は、ストアード・プロシージャーを呼び出してサーバー上でインポートを実行することができます。

ストアード・プロシージャーは、以下の場合には呼び出されません。

- アプリケーションとデータベースとでコード・ページが違っている場合。
- インポートされるファイルが、複数に分かれた PC/IXF ファイルである場合。
- データのインポートに使用される方式が列名か相対列位置のいずれかである場合。
- 指定されたターゲット列リストが 4 KB を超えている場合。
- LOBS FROM 節または `lobsinfile` 修飾子が指定されている場合。
- ASC ファイルに NULL INDICATORS 節が指定されている場合。

インポートでストアード・プロシージャーを使用する場合は、サーバーにインストールされているデフォルト言語を使ってメッセージ・ファイル内にメッセージが作成されます。クライアントとサーバーとで言語が同じ場合、メッセージはアプリケーションの言語になります。

インポート・ユーティリティーは、 `sqllib` ディレクトリー (または **DB2INSTPROF** レジストリー変数が指定されている場合はそれによって示されるディレクトリー) の `tmp` サブディレクトリーに、2 つの一時ファイルを作成します。 1 つのファイルはデータ用、もう 1 つのファイルはインポート・ユーティリティーが生成するメッセージ用です。

サーバー上でのデータの書き込みまたはオープンに関してエラーが出された場合は、以下の点を確認してください。

- ディレクトリーが存在する。
- それらのファイル用の十分のディスク・スペースがある。
- インスタンス所有者にそのディレクトリーでの書き込み許可がある。

インポート・ユーティリティーでサポートされている表ロッキング・モード

インポート・ユーティリティーは、2 つの表ロッキング・モードをサポートしています。つまり、オフラインの ALLOW NO ACCESS モードとオンラインの ALLOW WRITE ACCESS モードです。

`ALLOW NO ACCESS` モードでは、複数の同時アプリケーションから表データにアクセスできないようになります。`ALLOW WRITE ACCESS` モードでは、複数の同時アプリケーションからインポート・ターゲット表へ読み取りおよび書き込みアクセスするこ

とができます。モードを明示的に指定しない場合は、デフォルト・モードの ALLOW NO ACCESS でインポートが実行されます。さらに、インポート・ユーティリティーは、デフォルトで、分離レベル RS (読み取り固定) でデータベースにバインドされます。

オフライン・インポート (ALLOW NO ACCESS)

ALLOW NO ACCESS モードのインポートは、行を挿入する前にターゲット表の排他 (X) ロックを取得します。表のロックを保持することには、2 つの意味があります。

- 第 1 に、他のアプリケーションがインポート・ターゲット表に対する表ロックまたは行ロックを保持していると、インポート・ユーティリティーは、そのようなアプリケーションが変更内容をコミットするかロールバックするまで待機します。
- 第 2 に、インポートの実行中に、ロックを要求している他のすべてのアプリケーションは、そのインポート操作の完了を待機します。

注: locktimeout 値を指定すれば、インポート・ユーティリティーをはじめとするアプリケーションがロックをいつまでも待機しつづける事態を回避できます。
インポートは、操作の開始時に排他ロックを要求することによって、他のアプリケーションが処理中に同じターゲット表に対する行ロックを保持した場合に発生するデッドロックを回避します。

オンライン・インポート (ALLOW WRITE ACCESS)

ALLOW WRITE ACCESS モードのインポート・ユーティリティーは、ターゲット表の非排他 (IX) ロックを取得します。表に対してこのロックをかけることには 2 つの意義があります。

- 非互換の表ロックをかけている他のアプリケーションがあると、そのようなアプリケーションがすべて変更内容をコミットまたはロールバックするまでインポート・ユーティリティーはデータの挿入を開始しません。
- インポートが実行されているかぎり、非互換の表ロックを要求している他のすべてのアプリケーションは、そのインポートが現在のトランザクションをコミットするかまたはロールバックするまで待機します。インポートでの表ロックは、次のトランザクションに持ち越されることはないことに注意してください。これを受けて、オンライン・インポートでは、各コミットの完了ごとに表ロックの要求と、おそらくは待機を行う必要があります。
- 非互換の行ロックをかけている他のアプリケーションがある場合、そのようなアプリケーションがすべて変更内容をコミットまたはロールバックするまでインポート・ユーティリティーはデータの挿入を停止します。
- インポートの実行中に、非互換の行ロックを要求している他のすべてのアプリケーションは、そのインポート操作が現在のトランザクションをコミットするかロールバックするまで待機します。

ALLOW WRITE ACCESS インポートは、オンライン・プロパティーを維持しながら、デッドロックの可能性を減らすために、現在のトランザクションを周期的にコミットして行ロックをすべて解放することで、排他表ロックへのエスカレーションが発生しないようにしています。コミット頻度が明示的に設定されていなければ、インポ

ートは、COMMITCOUNT AUTOMATIC が指定されている場合と同じようにコミットを実行します。 COMMITCOUNT が 0 に設定されていると、コミットは実行されません。

ALLOW WRITE ACCESS モードには、以下のものとの互換性はありません。

- REPLACE、CREATE、REPLACE_CREATE のいずれかのモードのインポート
- バッファー挿入を使用したインポート
- ターゲット・ビューへのインポート
- 階層表へのインポート
- ロックの細分度が (ALTER TABLE ステートメントの LOCKSIZE パラメーターで) 表レベルに設定されている表へのインポート

ロード・ユーティリティー

ロードの概要

ロード・ユーティリティーを使用すれば、新しく作成した表やすでにデータが入っている表に、大量のデータを効率よく移動することができます。

このユーティリティーは、XML、ラージ・オブジェクト (LOB)、ユーザー定義タイプ (UDT) などのほとんどのデータ・タイプを処理できます。

インポート・ユーティリティーは SQL INSERT を実行するのに対し、ロード・ユーティリティーはフォーマット設定したページをデータベースに直接書き込むため、インポート・ユーティリティーよりも処理が高速です。

ロード・ユーティリティーはトリガーを起動したり、参照制約や表制約のチェックを実行したりしません (索引の一意性の妥当性チェックを除く)。

ロードのプロセスは、以下に示す 4 つの明確なフェーズ (段階) で構成されています (43 ページの図 3 を参照)。

1. **ロード**
   ロード・フェーズ中に、データは表にロードされ、必要に応じて索引キーと表統計が集められます。保管点または整合点は、**LOAD** コマンドの **SAVECOUNT** パラメーターによって指定されるインターバルで確立されます。保管点の時点で正常にロードされた入力行の数を示すメッセージが生成されます。
2. **構築**
   構築フェーズ中には、ロード・フェーズ中に収集した索引キーに基づいて索引が生成されます。索引キーはロード・フェーズ中にソートされ、索引統計が集められます (**STATISTICS USE PROFILE** オプションを指定してプロファイルが索引統計情報の収集を示す場合)。この統計データは、**RUNSTATS** コマンドによって収集される統計とよく似たものです。
3. **削除**
   削除フェーズ中に、ユニーク・キーまたは主キー違反の発生した行が表から除かれます。削除されるこれらの行はロード例外表に保管されます (表が指定された場合)。
4. **索引コピー**
   索引コピー・フェーズでは、SYSTEM TEMPORARY 表スペースから元の表スペ

ースに索引データがコピーされます。これは、**READ ACCESS** オプションを指定したロード操作時に、索引作成に SYSTEM TEMPORARY 表スペースが指定されていた場合にのみ発生します。

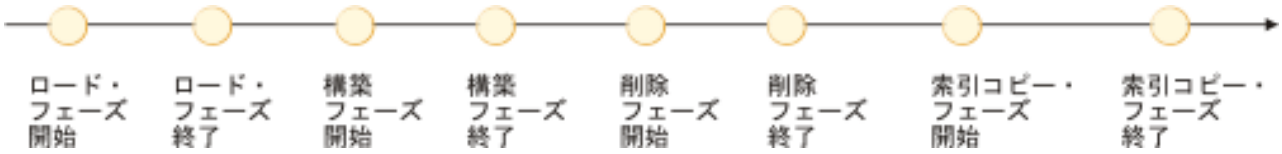


図 3. ロード・プロセスの 4 つのフェーズ (ロード、構築、削除、および索引コピー)

注: ロード・ユーティリティーを呼び出した後で **LIST UTILITIES** コマンドを使って、ロード操作の進行状況をモニターすることができます。

データをロードするには、以下の情報が必要になります。

- 入力となるファイル、Named PIPE、または装置のパスと名前。
- ターゲット表の名前または別名。
- 入力ソースのフォーマット。このフォーマットは、DEL、ASC、PC/IXF、または CURSOR です。
- 入力データを表に追加するか、それとも表内の既存データを置換するか。
- アプリケーション・プログラミング・インターフェース (API) db2Load によってユーティリティーが起動される場合、メッセージ・ファイル名。

ロード・モード

- **INSERT**
  このモードのロードは、既存のデータに変更を加えずに表に対して入力データを付加します。
- **REPLACE**
  このモードのロードは、表から既存のデータを削除して、表に入力データを追加します。
- **RESTART**
  このモードでは、中断したロードが再開されます。ほとんどの場合、ロードは失敗したフェーズから再開されます。失敗したフェーズがロード・フェーズだった場合、ロードは最後に成功した整合点から再開されます。
- **TERMINATE**
  このモードでは、失敗したロード操作がロールバックされます。

指定できるオプションは次のとおりです。

- ロードするデータがクライアントに置かれていること (リモート接続されたクライアントからロード・ユーティリティーを起動する場合)。 **CLIENT** オプションを指定したとしても、XML および LOB データは常にサーバーから読み取られることに注意してください。
- データのロードに使用する方式: 列のロケーション、列名、または相対列ロケーション。
- ユーティリティーが整合点を確立する頻度。
- データの挿入先となる表の列の名前。

- ロード操作の進行中に、表の中に事前に存在するデータを照会できるかどうか。
- ロード操作で、他のユーティリティーまたはアプリケーションが表の使用を終了するのを待機するようにするか、あるいは続行する前にその他のアプリケーションからの接続を強制的に切断するか。
- 索引を作成する代替 SYSTEM TEMPORARY 表スペース。
- LOB が格納されている入力ファイルのパスと名前。

  注: ロード・ユーティリティーは **COMPACT** LOB オプションを考慮しません。
- メッセージ・ファイル名。ロード操作においては、メッセージ・ファイルを作成するよう指定できます。メッセージ・ファイルには、それらの操作に関連したエラー、警告、および通知の各メッセージが入れられます。これらのファイルの名前は、MESSAGES パラメーターで指定します。

  注:

  1. メッセージ・ファイルの内容を表示できるのは、操作が終了した後でのみです。ロード操作の実行中にロード・メッセージを表示するには、**LOAD QUERY** コマンドを使用できます。
  2. メッセージ・ファイル内では、各メッセージはそれぞれ新たな行で始まり、DB2 メッセージ検索機能により提供される情報がそこに入れられます。
- ロードしている列値に暗黙の小数点があるかどうか。
- 表をロードした後、使用可能なフリー・スペースの大きさをユーティリティーが変更するかどうか。
- ロード・プロセス中に統計情報を収集するかどうか。このオプションは、ロード操作を REPLACE モードで実行する場合にのみサポートされます。統計は、表に定義されているプロファイルに従って収集されます。 **LOAD** コマンドの実行の前に、**RUNSTATS** コマンドを使用してこのプロファイルをあらかじめ作成しておく必要があります。プロファイルを作成していない場合に、プロファイルに従って統計を収集するようロード操作に指示すると、エラー・メッセージが戻されます。

  既存データが入っている表にロードする場合、統計情報は収集されません。追加がなされた表に関する現行の統計情報を収集するには、ロード・プロセスの完了後に **RUNSTATS** ユーティリティーを呼び出します。ユニーク索引のある表に関する統計情報を収集していて、削除フェーズ中に重複キーが削除された場合、それらのレコードの削除に合わせて統計情報が更新されることはありません。重複レコードの有効な数を入手する場合は、ロード操作中には統計情報を収集しないでください。その代わりに、ロード・プロセスの完了後に **RUNSTATS** ユーティリティーを呼び出すようにしてください。
- 変更部分のコピーを維持するかどうか。これにより、データベースのロールフォワード・リカバリーが可能になります。このオプションは、データベースのロールフォワード・リカバリーが無効になっている場合、つまりデータベース構成パラメーター **logarchmeth1** および **logarchmeth2** が OFF に設定されている場合にはサポートされません。コピーが作成されていないのにロールフォワード・リカバリーが有効になっていると、ロード操作の完了時に表スペースがバックアップ・ペンディング状態のままになります。

  データベースを完全にリカバリーできるようにするには、ログを記録する必要があります。ロード・ユーティリティーを利用した場合、データのロードに関連し

たロギングはほとんど不要です。ログを記録する代わりに、表のうちのロードした部分のコピーを作成することもできます。障害発生後にデータベースをリカバリーできるようになっているデータベース環境では、以下のいずれかを実行できます。

– 表のうちのロードした部分のコピーを作成することを明示的に要求する。
– 表の属する表スペースのバックアップを、ロード操作の完了直後に取る。

データベース構成パラメーター **logindexbuild** を設定し、しかも **COPY YES** リカバリー可能性オプションと INCREMENTAL 索引作成オプションを指定してロード操作を起動した場合、そのロードでは索引の修正がすべてログ記録されます。これらのオプションを使用する利点は、このロードのログ・レコードを先へ順にたどっていけば、索引のリカバリーも行われるという点にあります (これに対して、通常は、ロードで REBUILD 索引付けモードを使わないかぎり、索引はリカバリーされません)。

既存データが入っている表にロードする際、データベースがリカバリー可能でない場合は、ロード・ユーティリティーを呼び出す前に、そのデータベースあるいはロードする表の表スペースのバックアップ・コピーがあることを確認し、エラーがあってもリカバリーできるようにしておいてください。

リカバリー可能なデータベースで複数のロード操作を連続して実行する場合は、それぞれのロード操作をリカバリー不能に指定しておき、一連のロードの最後にバックアップを取ることにより、各ロード操作を **COPY YES** オプション付きで呼び出す場合よりも短い時間で一連の操作を実行できます。 NONRECOVERABLE オプションを使用すると、あるロード・トランザクションをリカバリー不能としてマークし、それ以降にロールフォワード操作を実行してもリカバリーできなくするよう指定できます。ロールフォワード・ユーティリティーはそのトランザクションをスキップし、データのロード先の表を "invalid (無効)" としてマークします。さらに、このユーティリティーは、それ以降のその表に対するトランザクションをすべて無視します。ロールフォワードの完了後、このような表はドロップすることしかできません (図 4 を参照)。このオプションを指定すると、表スペースはロード操作後にバックアップ・ペンディング状態になりません。また、ロードしたデータのコピーをロード操作中に作成する必要はありません。

図 4. ロールフォワード操作時のリカバリー不能処理

- ロード操作中の一時ファイル作成時に使用する完全修飾パス。パス名は、**LOAD** コマンドの **TEMPFILES PATH** パラメーターで指定します。デフォルト値はデータベースのパスです。このパスはサーバー・マシン上にあり、 DB2 のインスタンスが排他的にアクセスします。そのため、このパラメーターに指定するパス名の修

飾は、クライアントではなくサーバーのディレクトリー構造を反映していなければならず、 DB2 インスタンス所有者にはそのパスに対して読み取りと書き込みの両方の許可が必要です。

ロードを使用するために必要な特権と権限

ユーザーは、特権によってデータベース・リソースを作成したりアクセスしたりすることが可能になります。権限レベルは、特権をグループ化する手段となるものであり、さらに高水準のデータベース・マネージャーの保守およびユーティリティーのさまざまな操作を提供します。それらの働きにより、データベース・マネージャーとそのデータベース・オブジェクトへのアクセスが制御されます。ユーザーは、適切な許可 (必要な特権または権限) が付与されているオブジェクトにしかアクセスできません。

データを表にロードするには、以下のいずれかが必要です。

- DATAACCESS 権限
- データベースに対する LOAD 権限または DBADM 権限と次のいずれか
  - 表の INSERT 特権。INSERT モード、および (直前のロード INSERT 操作を終了するための) TERMINATE モード、もしくは (直前のロード INSERT 操作を再始動するための) RESTART モードでロード・ユーティリティーが呼び出される場合。
  - 表の INSERT および DELETE 特権。REPLACE モード、および (直前のロード REPLACE 操作を終了するための) TERMINATE モード、もしくは (直前のロード REPLACE 操作を再始動するための) RESTART モードでロード・ユーティリティーが呼び出される場合。
  - 例外表の INSERT 特権 (例外表をロード操作の一部として使用する場合)。
  - SYSCAT.TABLES の SELECT 特権は、LOAD がカタログ表を照会する場合に必要です。

すべてのロード・プロセス (および一般にすべての DB2 サーバー・プロセス) はインスタンス所有者が所有しており、これらのプロセスはすべてインスタンスの ID を使って必要なファイルにアクセスするため、インスタンス所有者にはデータ・ファイルを入力するために読み取りアクセスが必要です。だれがコマンドを呼び出すかに関係なく、これらの入力データ・ファイルはインスタンス所有者から読み取り可能になっていなければなりません。

REPLACE オプションを指定する場合、セッション許可 ID には表をドロップするための権限が必要です。

Windows、およびWindows.NET オペレーティング・システムで、DB2 が Windows サービスとして実行されている場合、ネットワーク・ドライブに常駐するファイルからデータをロードするには、これらのファイルに対する読み取りアクセスを持つユーザー・アカウントの下で実行するようにDB2 のサービスを構成する必要があります。

注:

- 保護列を持つ表にデータをインポートするには、表内のすべての保護列への書き込みアクセスを可能にする LBAC 信用証明情報がセッション許可 ID に必要です。

- 保護行を持つ表にデータをロードするには、表を保護するセキュリティー・ポリシーの一部である、書き込みアクセス用のセキュリティー・ラベルが、セッション許可 ID に対して付与されていなければなりません。

LOAD 権限

データベース・レベルの LOAD 権限、および表に対する INSERT 特権を持っているユーザーは、 **LOAD** コマンドを使用してデータを表にロードすることができます。

注: DATAACCESS 権限を保有していると、LOAD コマンドへの全アクセス権限がユーザーに付与されます。

データベース・レベルの LOAD 権限、および表に対する INSERT 特権を持っているユーザーは、直前のロード操作でデータを挿入するロードを行った場合に、 **LOAD RESTART** または **LOAD TERMINATE** を行うことができます。

データベース・レベルの LOAD 権限、および表に対する INSERT 特権と DELETE 特権を持っているユーザーは、 **LOAD REPLACE** コマンドを使用できます。

直前のロード操作でロード置換を行った場合、ユーザーは、DELETE 特権が付与されていないと、 **LOAD RESTART** または **LOAD TERMINATE** を行うことができません。

ロード操作の一部として例外表が使用される場合、ユーザーには、その例外表に対する INSERT 特権が必要です。

この権限を持っているユーザーは、 **QUIESCE TABLESPACES FOR TABLE**、 **RUNSTATS**、および **LIST TABLESPACES** コマンドを実行することができます。

データのロード

ロード・ユーティリティーは、新しく作成した表や既にデータが入っている表に、大量のデータを効率よく移動します。

始める前に

ロード・ユーティリティーを起動するには、その前にデータのロード先となるデータベースに接続されているか、または暗黙接続が可能な状態になっていなければなりません。

ユーティリティーは COMMIT ステートメントを発行するので、ロード・ユーティリティーの呼び出し前に COMMIT または ROLLBACK ステートメントを発行して、すべてのトランザクションを完了し、すべてのロックを解除しておいてください。

データは入力ファイルに示されている順序でロードされます。ただしマルチディメンション・クラスタリング (MDC) 表、パーティション表、または **anyorder** ファイル・タイプ修飾子を使用する場合を除きます。特定の順序を望む場合には、ロード操作を試行する前にデータをソートしてください。クラスタリングする必要がある場合は、ロードする前にクラスタリング索引に従ってデータをソートしておいてください。マルチディメンション・クラスタリング (MDC) 表にデータをロードする際には、ロード操作前のソートは必要ありません。データは MDC 表定義に従っ

てクラスター化されます。パーティション表にデータをロードする際には、ロード操作前のソートは必要ありません。データは表定義に従ってパーティション化されます。

制約事項

ロード・ユーティリティーに適用される制約事項がいくつかあります。

- ニックネームへのデータのロードはサポートされていません。
- 型付き表または構造化タイプ列をもつ表へのデータのロードはサポートされていません。
- 宣言済み一時表および作成済み一時表へのデータのロードはサポートされていません。
- XML データはサーバー・サイドからのみ読み取ることができます。クライアントから XML ファイルを読み取る場合、インポート・ユーティリティーを使用します。
- バックアップ・ペンディング状態にある表スペースで表を作成またはドロップすることはできません。
- DB2 Connect、あるいは DB2 バージョン 2 より前のサーバー・レベルを経由してアクセスされるデータベースにデータをロードすることはできません。現行でのみ利用可能なオプションは、旧リリースのサーバーでは使用できません。
- **LOAD REPLACE** 操作中にエラーが発生すると、表のオリジナル・データは失われます。ロード操作を再開できるようにするには、入力データのコピーを作成しておいてください。
- 新しくロードした行ではトリガーが起動されません。トリガーに関連付けられたビジネス・ルールは、ロード・ユーティリティーによって適用されません。
- 暗号化されたデータのロードは、サポートされていません。

パーティション表にロードする際には、ロード・ユーティリティーにいくつかの制約事項が適用されます。

- パーティション・エージェントが複数存在する場合、整合点はサポートされません。
- データ・パーティションのサブセットにデータをロードしている間、その他のデータ・パーティションを完全にオンラインにしておく機能はサポートされません。
- ロード操作または SET INTEGRITY ペンディング操作で使用される例外表は、パーティション化できません。
- ロード・ユーティリティーが挿入モードまたは再始動モードで実行されていて、デタッチされた従属データがロード・ターゲット表にある場合には、ユニーク索引を再作成することはできません。

手順

ロード・ユーティリティーを起動するには、次のようにします。

- コマンド行プロセッサー (CLP) で **LOAD** コマンドを発行します。
- クライアント・アプリケーションから db2Load アプリケーション・プログラミング・インターフェース (API) を呼び出します。

- **LOAD** コマンドに関する IBM Data Studio のタスク・アシストを開きます。

例

CLP によって発行する **LOAD** コマンドの例を以下に示します。

```
db2 load from stafftab.ixf of ixf messages staff.msgs
   insert into userid.staff copy yes use tsm data buffer 4000
```

この例の詳細は次のとおりです。

- 警告メッセージやエラー・メッセージはすべて staff.msgs ファイルに入れられます。
- 変更された部分のコピーは、Tivoli® Storage Manager (TSM) に保管されます。
- 4,000 ページ分のバッファー・スペースがロード操作中に使用されます。

CLP によって発行する **LOAD** コマンドの別の例を以下に示します。

```
db2 load from stafftab.ixf of ixf messages staff.msgs
   tempfiles path /u/myuser replace into staff
```

この例の詳細は次のとおりです。

- 表データは置換されます。
- **TEMPFILES PATH** パラメーターを使用して、一時ファイルを書き込むサーバー・パスとして /u/myuser を指定しています。

注: これらの例では、ロード用入力ファイルに相対パス名を使っています。相対パス名を使用できるのは、データベースと同じデータベース・パーティション上にあるクライアントから呼び出す場合だけです。できれば完全修飾パス名を使用してください。

次のタスク

ロード・ユーティリティーを呼び出した後で **LIST UTILITIES** コマンドを使って、ロード操作の進行状況をモニターすることができます。 **INSERT** モード、**REPLACE** モード、または **RESTART** モードのいずれかでロード操作を実行する場合、進行状況の詳細モニタリングのサポートを利用することができます。現在のロード・フェーズに関する詳細情報を表示するには、**SHOW DETAILS** パラメーターを指定して **LIST UTILITIES** コマンドを出します。**TERMINATE** モードで実行するロード操作では、詳細情報は得られません。 **LIST UTILITIES** コマンドは、ロード終了ユーティリティーが現在稼働中であることを示すだけです。

ロード操作は、ユニーク制約、パーティション表の範囲制約、生成される列、および LBAC セキュリティー規則を保守します。他のすべての制約では、表は、ロード操作の開始時点に「SET INTEGRITY ペンディング」状態に置かれます。ロード操作が完了したら、SET INTEGRITY ステートメントを使って、表の「SET INTEGRITY ペンディング」状態を終了しなければなりません。

XML データのロード

ロード・ユーティリティーを使用して、大量の XML データを表に効率的に移動できます。

データを XML 表列にロードする場合、XML FROM オプションを使用して入力 XML データ (1 つまたは複数) のパスを指定できます。例えば、データを XML ファイル /home/user/xmlpath/xmlfile1.xml からロードするには、以下のコマンドを使用できます。

```
LOAD FROM data1.del OF DEL XML FROM /home/user/xmlpath INSERT INTO USER.T1
```

区切られた ASCII 入力ファイル data1.del には、ロードする XML データの場所を説明する XML データ指定子 (XDS) が含まれます。例えば、以下の XDS は、ファイル xmldata.ext 内のオフセット 123 バイトにある、長さが 456 バイトの XML 文書について説明しています。

```
<XDS FIL='xmldata.ext' OFF='123' LEN='456' />
```

宣言済みカーソルを使用した XML データのロードがサポートされています。以下の例ではカーソルを宣言し、そのカーソルと **LOAD** コマンドを使用して、表 CUSTOMERS のデータを表 LEVEL1_CUSTOMERS に追加します。

```
DECLARE cursor_income_level1 CURSOR FOR
   SELECT * FROM customers
   WHERE XMLEXISTS('$DOC/customer[income_level=1]');

LOAD FROM cursor_income_level1 OF CURSOR INSERT INTO level1_customers;
```

XML データを XML 列にロードする際に、**LOAD** コマンドの ANYORDER ファイル・タイプ修飾子がサポートされています。

ロードの際に、分散統計はタイプ XML の列については収集されません。

パーティション・データベース環境での XML データのロード

データベース・パーティション間で分散される表の場合、複数の XML データ・ファイルから XML データを表に並行してロードできます。XML データをファイルから表にロードする場合、ロードが実行されるすべてのデータベース・パーティションで XML データ・ファイルが読み取り可能でなければなりません。

スキーマに対して挿入された文書を検証する

XMLVALIDATE オプションにより、 XML 文書がロードされるときに、それらを XML スキーマに対して妥当性検査できます。次の例では、区切られた ASCII 入力ファイル data2.del 内で XDS によって識別されるスキーマに対して、着信 XML 文書が検証されます。

```
LOAD FROM data2.del OF DEL XML FROM /home/user/xmlpath XMLVALIDATE
     USING XDS INSERT INTO USER.T2
```

この場合、XDS には XML スキーマの完全修飾 SQL ID "S1.SCHEMA_A" のある SCH 属性が妥当性検査で使用するために含まれます。

```
<XDS FIL='xmldata.ext' OFF='123' LEN='456' SCH='S1.SCHEMA_A' />
```

構文解析オプションの指定

XMLPARSE オプションを使用して、ロードされた XML 文書の空白文字を保存するかストリップするかを指定できます。次の例では、SQL ID "S2.SCHEMA_A" のあるスキーマに対してすべてのロードされた XML 文書が検証されて、これらの文書は空白文字を保存しながら構文解析されます。

```
LOAD FROM data2.del OF DEL XML FROM /home/user/xmlpath XMLPARSE PRESERVE
WHITESPACE XMLVALIDATE USING SCHEMA S2.SCHEMA_A INSERT INTO USER.T1
```

ロード・セッション - CLP の例

例 1

TABLE1 には以下に示す 5 つの列があります。

- COL1 VARCHAR 20 NOT NULL WITH DEFAULT
- COL2 SMALLINT
- COL3 CHAR 4
- COL4 CHAR 2 NOT NULL WITH DEFAULT
- COL5 CHAR 2 NOT NULL

ASCFILE1 には以下に示す 6 つのエレメントがあります。

- ELE1、位置 01 から 20
- ELE2、位置 21 から 22
- ELE3、位置 23 から 23
- ELE4、位置 24 から 27
- ELE5、位置 28 から 31
- ELE6、位置 32 から 32
- ELE7、位置 33 から 40

データ・レコード:

```
1...5...10...15...20...25...30...35...40
Test data 1         XXN 123abcdN
Test data 2 and 3   QQY    XXN
Test data 4,5 and 6 WWN6789    Y
```

以下に示すコマンドで、ファイルから表をロードします。

```
db2 load from ascfile1 of asc modified by striptblanks reclen=40
   method L (1 20, 21 22, 24 27, 28 31)
   null indicators (0,0,23,32)
   insert into table1 (col1, col5, col2, col3)
```

注:

1. MODIFIED BY パラメーターで `striptblanks` を指定すると、 VARCHAR 列の中のブランクが切り捨てられるようになります (例えば、行 1、2、および 3 の長さがそれぞれ 11、17、および 19 バイトである COL1)。
2. MODIFIED BY パラメーターに `reclen=40` を指定することにより、各入力レコードの末尾には改行文字がなく、各レコードの長さが 40 バイトであることを示しています。最後の 8 バイトは、表のロードには使用されません。
3. COL4 は入力ファイルにはないので、そのデフォルト値 (NOT NULL WITH DEFAULT と定義されている) を使用して TABLE1 に挿入されます。
4. 位置 23 および 32 は、TABLE1 の COL2 と COL3 に NULL をロードするかどうかを行ごとに示すために使用されます。特定のレコードのうち列の NULL 標識位置が `Y` なら、その列は NULL になります。それが `N` の場合は、入力レコードのその列のデータ位置 (L(.....) で定義) にあるデータ値が、その行の列

データのソースとして使用されます。この例では、行 1 のどの列も NULL ではなく、行 2 の COL2 は NULL であり、行 3 の COL3 は NULL です。

5. この例では、COL1 と COL5 の NULL INDICATORS は 0 (ゼロ) として指定されますが、それはそのデータを NULL 不可能であることを示しています。
6. 特定の列に対する NULL INDICATOR は入力レコードのどの位置でも可能ですが、その位置は必ず指定しなければならず、 `Y` または `N` のいずれかの値が提供される必要があります。

例 2 (ダンプ・ファイルの使用)

表 FRIENDS は以下のように定義されています。

```
table friends "( c1 INT NOT NULL, c2 INT, c3 CHAR(8) )"
```

この表に対して、以下に示すデータ・レコードのロードを試みたとします。

```
23, 24, bobby
, 45, john
4,, mary
```

第 2 行は最初の INT が NULL であり、列定義では NOT NULL が指定されているためにリジェクトされます。 DEL フォーマットと互換でない開始文字の入った列は、エラーを生成し、レコードはリジェクトされます。そのようなレコードはダンプ・ファイルに書き込まれます。

区切り文字の外側の列の中にある DEL データは無視されますが、警告が生成されます。例えば、以下のようにします。

```
22,34,"bob"
24,55,"sam" sdf
```

ユーティリティーはこの表の第 3 列にある "sam" をロードし、文字 "sdf" には警告のフラグが付けられます。このレコードはリジェクトされません。別の例として、

```
22 3, 34,"bob"
```

ユーティリティーは `22,34,"bob"` をロードし、列 1 の `22` の後にある一部のデータが無視されたという警告を生成します。このレコードはリジェクトされません。

例 3 (ID 列がある表のロード)

TABLE1 には、以下の 4 つの列があります。

- C1 VARCHAR(30)
- C2 INT GENERATED BY DEFAULT AS IDENTITY
- C3 DECIMAL(7,2)
- C4 CHAR(1)

TABLE2 は、C2 が GENERATED ALWAYS ID 列であることを除き、 TABLE1 と同じです。

DATAFILE1 内のデータ・レコード (DEL フォーマット):

```
"Liszt"
"Hummel",,187.43, H
"Grieg",100, 66.34, G
"Satie",101, 818.23, I
```

DATAFILE2 内のデータ・レコード (DEL フォーマット):

```
"Liszt", 74.49, A
"Hummel", 0.01, H
"Grieg", 66.34, G
"Satie", 818.23, I
```

注:

1. 以下のコマンドでは、行 1 および 2 の ID 値が生成されます。これは、DATAFILE1 内にこれらの行の ID 値が存在しないためです。ただし、行 3 および 4 には、ユーザー提供の ID 値である 100 および 101 がそれぞれ割り当てられます。

   ```
   db2 load from datafile1.del of del replace into table1
   ```

2. DATAFILE1 を TABLE1 にロードして、すべての行の ID 値が生成されるようにするには、以下のコマンドのいずれかを発行してください。

   ```
   db2 load from datafile1.del of del method P(1, 3, 4)
      replace into table1 (c1, c3, c4)
   db2load from datafile1.del of del modified by identityignore
      replace into table1
   ```

3. DATAFILE2 を TABLE1 にロードして、それぞれの行ごとに ID 値が生成されるようにするには、以下のコマンドのいずれかを発行してください。

   ```
   db2 load from datafile2.del of del replace into table1 (c1, c3, c4)
   db2 load from datafile2.del of del modified by identitymissing
      replace into table1
   ```

4. DATAFILE1 を TABLE2 にロードして、 ID 値である 100 および 101 が行 3 および 4 にそれぞれ割り当てられるようにするには、以下のコマンドを発行してください。

   ```
   db2 load from datafile1.del of del modified by identityoverride
      replace into table2
   ```

   この場合、行 1 および 2 はリジェクトされます。これは、ユーティリティーへの指示により、ユーザー提供の値を優先し、システムが生成した ID 値をオーバーライドするようになっているためです。ユーザー提供の値が存在しない場合でも、 ID 列が暗黙的に非 NULL であるため、この行はリジェクトする必要があります。

5. 識別に関係するファイル・タイプ修飾子を使用せずに DATAFILE1 を TABLE2 にロードすると、行 1 と 2 はロードされますが、行 3 と 4 はリジェクトされます。これは、行 3 と 4 では独自に非 NULL 値が提供されており、 ID 列が GENERATED ALWAYS であるためです。

例 3 (CURSOR からのロード)

MY.TABLE1 には次の 3 つの列があります。

- ONE INT
- TWO CHAR(10)
- THREE DATE

MY.TABLE2 には次の 3 つの列があります。

- ONE INT
- TWO CHAR(10)
- THREE DATE

カーソル MYCURSOR は以下のように定義されます。

```
declare mycursor cursor for select * from my.table1
```

次のコマンドは、すべてのデータを MY.TABLE1 から MY.TABLE2 にロードします。

```
load from mycursor of cursor method P(1,2,3) insert into
  my.table2(one,two,three)
```

注:

1. 単一の LOAD コマンドには、カーソル名を 1 つだけ指定できます。つまり、`load from mycurs1, mycurs2 of cursor...` は許可されません。
2. カーソルからのロードに有効な METHOD 値は、`P` および `N` だけです。
3. この例では、METHOD P および挿入列リスト `(one,two,three)` はデフォルト値を示すため、これらを省略してもかまいません。
4. MY.TABLE1 は、表、ビュー、別名、またはニックネームが使用できます。

パーティション表におけるロードの考慮事項

以下の一般制約事項を除き、既存のすべてのロード・フィーチャーはターゲット表がパーティション化されている場合にサポートされます。

- パーティション・エージェントが複数存在する場合、整合点はサポートされません。
- データ・パーティションのサブセットにデータをロードしている間、その他のデータ・パーティションを完全にオンラインのままにしておく機能はサポートされません。
- ロード操作で使用される例外表は、パーティション化できません。
- ターゲット表に XML 列がある場合、例外表を指定することはできません。
- ロード・ユーティリティーが挿入モードまたは再始動モードで実行されていて、デタッチされた従属データがロード・ターゲット表にある場合には、ユニーク索引を再作成することはできません。
- MDC 表のロードと同様、入力データ・レコードの厳密な順序は、パーティション表をロードする際には保持されません。順序はセルまたはデータ・パーティションの中のみで維持されます。
- 各データベース・パーティションで複数のフォーマッターを使用するロード操作では、入力レコードの大まかな順序のみを保持します。各データベース・パーティション上で単一のフォーマッターを実行すると、入力レコードがセルまたは表パーティション・キーごとにグループ化されます。各データベース・パーティション上で単一のフォーマッターを実行するには、明示的に `CPU_PARALLELISM` に 1 を要求してください。

一般的なロードの動作

ロード・ユーティリティーは、データ・レコードを適切なデータ・パーティションに挿入します。ロードの前に入力データをパーティション化するための外部ユーティリティー (スプリッターなど) を使用する上での要件はありません。

ロード・ユーティリティーは、アタッチまたはデタッチされたデータ・パーティションにアクセスしません。データは可視のデータ・パーティションのみに挿入されます。可視のデータ・パーティションは、アタッチされたりデタッチされたりしません。また、ロード置換操作では、アタッチまたはデタッチされたデータ・パーティションを切り捨てることはありません。ロード・ユーティリティーではカタログ・システム表上のロックを獲得するため、ロード・ユーティリティーはコミットされていない ALTER TABLE トランザクションがあれば待機します。そのようなトランザクションは、カタログ表内の関連する行の排他ロックを獲得します。排他ロックを終了しなければロード操作は進行できません。これは、ロード操作の実行中は、コミットされていない ALTER TABLE ...ATTACH、DETACH、または ADD PARTITION トランザクションはありえないということを意味します。アタッチまたはデタッチされたデータ・パーティションに宛てられたすべての入力ソース・レコードはリジェクトされ、例外表が指定されている場合にはそこから取得できます。ターゲット表データ・パーティションの一部がアタッチまたはデタッチされた状態であったことを示すため、通知メッセージがメッセージ・ファイルに書き込まれます。ターゲット表に対応するカタログ表の関連する行のロックは、ロード・ユーティリティーの実行中に ALTER TABLE ...ATTACH、DETACH、または ADD PARTITION 操作を実行することによりユーザーがターゲット表のパーティションを変更することを防ぎます。

無効な行の処理

ロード・ユーティリティーで可視のデータ・パーティションのいずれにも属さないレコードが検出されると、そのレコードはリジェクトされ、ロード・ユーティリティーは処理を継続します。範囲制約違反のためにリジェクトされたレコードの数は明示的には表示されませんが、リジェクトされたレコードの全体数には含められます。範囲違反のためにレコードをリジェクトしても行の警告数は増加しません。範囲違反が検出されたものの、レコードごとのメッセージはログに記録されないということを示す単一のメッセージ (SQL0327N) がロード・ユーティリティーのメッセージ・ファイルに書き込まれます。例外表には、ターゲット表のすべての列に加えて、特定の行で発生した違反のタイプを記述する列が含まれます。無効データを含む行 (パーティション化できないデータを含む) は、ダンプ・ファイルに書き込まれます。

例外表への挿入は非効率であるため、どの制約違反を例外表に挿入するかを制御できます。例えば、ロード・ユーティリティーのデフォルトの動作は、範囲制約違反またはユニーク制約違反のためにリジェクトされた (その違反がなければ有効だった) 行を例外表に挿入することです。この動作は、FOR EXCEPTION 節を使用し、NORANGEEXC (範囲制約違反の場合) または NOUNIQUEEXC (ユニーク制約違反の場合) を指定することによってオフにすることができます。それらの制約違反を例外表に挿入しないことを指定する場合、または例外表を指定しない場合、範囲制約またはユニーク制約に違反する行に関する情報は失われます。

履歴ファイル

ターゲット表がパーティション化されている場合、対応する履歴ファイルの項目は、ターゲット表により範囲を設定された表スペースのリストを含みません。操作対象のオブジェクト ID (「T」ではなく「R」) は、ロード操作がパーティション表に対して実行されたことを示します。

ロード操作の終了

ロード置換を終了すると、すべてのデータ・パーティションが完全に切り捨てられ、ロード挿入を終了すると、すべてのデータ・パーティションがロード前の長さに切り捨てられます。ロード・コピー・フェーズで失敗した ALLOW READ ACCESS ロード操作の終了中に索引は無効になります。索引にタッチした ALLOW NO ACCESS ロード操作を終了する時にも索引は無効になります (それが無効になるのは、索引付けモードが再作成されているか増分保守の間にキーが挿入されたために索引が不整合状態になっているためです)。データを複数のターゲットにロードしても、ロード・フェーズ中に取られた整合点からロード操作を再始動できない点を除き、ロード・リカバリー操作に何の影響もありません。この場合、ターゲット表がパーティション化されている場合には、SAVECOUNT ロード・オプションは無視されます。この動作は、MDC ターゲット表へのデータのロードと一貫しています。

生成列

生成される列がパーティション・キー、ディメンション・キー、または分散キーのいずれかにある場合、 generatedoverride ファイル・タイプ修飾子は無視され、ロード・ユーティリティーは generatedignore ファイル・タイプ修飾子が指定された場合のように値を生成します。ここで不正な生成列値をロードすると、レコードが不適切な物理ロケーション (例えば不適切なデータ・パーティション、MDC ブロック、またはデータベース・パーティション) に配置されてしまう可能性があります。例えば、あるレコードがいったん間違ったデータ・パーティションに置かれると、SET INTEGRITY ではそのレコードを別の物理ロケーションに移動しなければなりませんが、それはオンラインでの SET INTEGRITY 操作中には行えません。

データの可用性

現行の ALLOW READ ACCESS ロード・アルゴリズムは、パーティション表に拡張されています。ALLOW READ ACCESS ロード操作の場合は、複数のリーダーが同時に表全体 (ロードするデータ・パーティションとロードしないものの両方を含む) にアクセスすることができます。

重要: バージョン 10.1 フィックスパック 1 以降、ALLOW READ ACCESS パラメーターは非推奨となっており、将来のリリースで除去される可能性があります。詳しくは、「*DB2 バージョン 10.1 の新機能*」の『LOAD コマンドの ALLOW READ ACCESS パラメーターが非推奨になった』を参照してください。

また、INGEST ユーティリティーはパーティション表をサポートしていて、LOAD コマンドで ALLOW READ ACCESS パラメーターを指定するよりもこのユーティリティーの方が、データ並行性と可用性に適しています。ターゲット表をロックすることなく、ファイルおよびパイプから大量のデータ

を移動することができます。また、経過時間または行数に基づいて、コミット後すぐにデータはアクセス可能になります。

データ・パーティションの状態

ロードに成功した後、特定の条件下では、可視のデータ・パーティションの表の状態が「SET INTEGRITY ペンディング」または「読み取りアクセスのみ」のいずれかまたは両方に変更される場合があります。ロード操作で保守できない制約が表にある場合に、データ・パーティションはこれらの状態になる可能性があります。そのような制約には、チェック制約とデタッチされたマテリアライズ照会表が含まれる場合があります。ロード操作に失敗すると、すべての可視のデータ・パーティションの表の状態が「ロード・ペンディング」になります。

エラー分離

データ・パーティション・レベルでのエラー分離はサポートされていません。エラーを分離するとは、エラーにならなかったデータ・パーティションでロードを継続し、エラーになったデータ・パーティションで停止するということを意味します。エラーは異なるデータベース・パーティションの間で分離できますが、ロード・ユーティリティーは可視のデータ・パーティションのサブセット上でトランザクションをコミットしたり、残りの可視のデータ・パーティションをロールバックしたりすることはできません。

その他の考慮事項

- いずれかの索引が無効とマークされている場合には増分索引付けはサポートされません。索引の再作成が必要な場合、またはデタッチされた従属物が SET INTEGRITY ステートメントでの妥当性検査を必要としている場合には、索引は無効であると見なされます。
- 範囲別パーティション化、ハッシュによる分散、またはディメンションによる編成のいずれかのアルゴリズムの組み合わせを使用してパーティション化された表へのロードもサポートされています。
- ロードの影響を受けるオブジェクトと表スペース ID のリストが含まれるログ・レコードの場合、これらのログ・レコード (LOAD START および COMMIT (PENDING LIST)) のサイズは非常に大きくなる場合があり、そうなると、他のアプリケーションで使用できるアクティブ・ログ・スペースの量が減少してしまいます。
- 表がパーティション化されていて、かつ分散されている場合、パーティション・データベースのロードがすべてのデータベース・パーティションには影響を与えない場合があります。出力データベース・パーティション上のオブジェクトのみが変更されます。
- ロード操作中、パーティション表のメモリー使用量は表の数とともに増加します。増加の合計は直線的にならない点に注意してください。データ・パーティションの数に比例するのはメモリー所要量全体のうちのほんの僅かだからです。

LBAC で保護されたデータのロードに関する考慮事項

保護行が含まれている表へのロード操作を正常に実行するには、LBAC (ラベル・ベースのアクセス制御) 信用証明情報が必要です。さらに、ターゲット表に関連付け

られているセキュリティー・ポリシーで有効なセキュリティー・ラベル、または有効なセキュリティー・ラベルに変換できるセキュリティー・ラベルを用意することも必要です。

有効な LBAC 信用証明情報がなければ、ロードは失敗し、エラー (SQLSTATE 42512) が返されます。入力データにセキュリティー・ラベルが含まれていない場合や、セキュリティー・ラベルが内部のバイナリー・フォーマットでない場合は、いくつかのファイル・タイプ修飾子を使用して、ロード操作を進めることができます。

保護行が含まれている表にデータをロードする場合は、ターゲット表にデータ・タイプ DB2SECURITYLABEL の列が 1 つ含まれています。入力データ行にその列の値が含まれていなければ、その行はリジェクトされます。ただし、ロード・コマンドに usedefaults ファイル・タイプ修飾子が指定されている場合は例外です。その場合に、その表を保護しているセキュリティー・ポリシーで書き込みアクセスが認められているセキュリティー・ラベルがあれば、そのセキュリティー・ラベルが使用されます。書き込みアクセスが認められているセキュリティー・ラベルがなければ、その行はリジェクトされ、処理が次の行に移ります。

保護行が含まれている表にデータをロードする場合に、入力データにデータ・タイプ DB2SECURITYLABEL の列の値が含まれていれば、その表にデータを挿入する場合と同じ規則が当てはまります。ロード対象の行を保護しているセキュリティー・ラベル (データ・ファイル内のその行にあるセキュリティー・ラベル) が書き込み可能であれば、その行の保護のためにそのセキュリティー・ラベルが使用されます。(つまり、そのセキュリティー・ラベルがデータ・タイプ DB2SECURITYLABEL の列に書き込まれます。)そのセキュリティー・ラベルで保護されている行への書き込みができない場合の動作は、ソース表を保護しているセキュリティー・ポリシーの作成方法によって異なります。

- ポリシーを作成した CREATE SECURITY POLICY ステートメントにオプション RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL が含まれていた場合には、行はリジェクトされます。
- CREATE SECURITY POLICY ステートメントにそのオプションが組み込まれていなかった場合や、 OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL オプションが代わりに組み込まれていた場合は、データ・ファイルに含まれているその行のためのセキュリティー・ラベルは無視されます。書き込みアクセスが認められているセキュリティー・ラベルがあれば、その行の保護のためにそのセキュリティー・ラベルが使用されます。この場合、エラーや警告は出されません。書き込みアクセスが認められているセキュリティー・ラベルがなければ、その行はリジェクトされ、処理が次の行に移ります。

区切り文字に関する考慮事項

データ・タイプ DB2SECURITYLABEL の列にデータをロードする場合、デフォルトでは、データ・ファイル内の値は、そのセキュリティー・ラベルの内部表記を構成する実際のバイトと見なされます。ただし、生データには改行文字が含まれている場合があり、その場合、**LOAD** コマンドは、その改行文字を行の区切り文字と誤解してしまう可能性があります。この問題がある場合は、行区切り文字よりもストリング区切り文字を優先するために、delprioritychar ファイル・タイプ修飾子を使用します。delprioritychar を使用すると、ストリング区切り文字の中にあるレコード区切り文字または

列区切り文字は、区切り文字と見なされなくなります。delprioritychar ファイル・タイプ修飾子は、改行文字が含まれている値が存在しない場合に使用しても差し支えありません。ただし、ロードの速度は少し低下します。

ロード対象のデータが ASC 形式の場合は、ロードされたセキュリティー・ラベルやセキュリティー・ラベル名に末尾空白が組み込まれる事態を回避するために、追加の手順を実行する必要があります。ASCII フォーマットでは、列位置が区切りとして使用されるので、可変長フィールドにロードするときにこの問題が発生する可能性があります。末尾ブランク・スペースを切り捨てるには、striptblanks ファイル・タイプ修飾子を使用します。

非標準のセキュリティー・ラベル値

セキュリティー・ラベルの値が、セキュリティー・ラベルの各コンポーネントの値を含んだストリングになっている場合、例えば、*S:(ALPHA,BETA)* のようになっている場合でも、データ・ファイルをロードできます。そのためには、ファイル・タイプ修飾子 seclabelchar を使用する必要があります。seclabelchar を使用すると、データ・タイプ DB2SECURITYLABEL の列の値は、セキュリティー・ラベル用のストリング・フォーマットで記述されたセキュリティー・ラベルを含む、ストリング定数と見なされます。ストリングが正しい形式でなければ、その行は挿入されず、警告 (SQLSTATE 01H53) が返されます。そのストリングが、表を保護しているセキュリティー・ポリシーに組み込まれている有効なセキュリティー・ラベルを記述したものでない場合も、その行は挿入されず、警告 (SQLSTATE 01H53) が返されます。

セキュリティー・ラベル列の値がセキュリティー・ラベル名になっている場合も、データ・ファイルをロードできます。この種のファイルをロードするには、ファイル・タイプ修飾子 seclabelname を使用する必要があります。seclabelname を使用すると、データ・タイプ DB2SECURITYLABEL の列のすべての値は、既存のセキュリティー・ラベルの名前を含んだストリング定数と見なされます。表を保護しているセキュリティー・ポリシーでその名前のセキュリティー・ラベルが存在しなければ、その行はロードされず、警告 (SQLSTATE 01H53) が返されます。

リジェクトされた行

ロード中にリジェクトされた行は、リジェクトされた理由に応じて、ダンプ・ファイルか例外表のいずれかに送られます (それらが **LOAD** コマンドで指定された場合)。構文解析エラーが原因でリジェクトされた行はダンプ・ファイルに送られます。セキュリティー・ポリシーを違反した行は例外表に送られます。

注: ターゲット表に XML 列がある場合、例外表を指定することはできません。

例

すべての例で、入力データ・ファイル myfile.del は DEL 形式になっています。すべての場合のデータのロード先は、以下のステートメントで作成された REPS という名前の表です。

```
create table reps (row_label db2securitylabel,
id integer,
name char(30))
security policy data_access_policy
```

次の例の入力ファイルには、デフォルト形式のセキュリティー・ラベルが含まれていると見なされます。

```
db2 load from myfile.del of del modified by delprioritychar insert into reps
```

次の例の入力ファイルには、セキュリティー・ラベル・ストリング・フォーマットのセキュリティー・ラベルが含まれていると見なされます。

```
db2 load from myfile.del of del modified by seclabelchar insert into reps
```

次の例の入力ファイルには、セキュリティー・ラベル列のセキュリティー・ラベル名が含まれていると見なされます。

```
db2 load from myfile.del of del modified by seclabelname insert into reps
```

ID 列のロードに関する考慮事項

ロード・ユーティリティーを使用すると、入力データに ID 列の値があるかどうかにかかわりなく、ID 列が含まれている表にデータをロードできます。

ID 関連のファイル・タイプ修飾子が使用されない場合、このユーティリティーは次のような規則に従って動作します。

- ID 列が GENERATED ALWAYS の場合、入力ファイル内の対応する行の中に ID 列用の値がないか、または NULL 値が明示的に指定されているときに、表の行用の ID 値が生成されます。 ID 列に非 NULL 値を指定すると、その行はリジェクトされます (SQL3550W)。
- ID 列が GENERATED BY DEFAULT の場合、ユーザー提供値が指定されていれば、ロード・ユーティリティーはその値を使用します。データが欠落しているかまたは明示的に NULL であれば、値が生成されます。

ロード・ユーティリティーは、ユーザー提供の ID 値に関して、 ID 列のデータ・タイプ (SMALLINT、INT、BIGINT、または DECIMAL) の値に対して通常以外の妥当性検査は行いません。値が重複していても報告されません。

ほとんどの場合、ロード・ユーティリティーは、行への ID 列値の割り当て順がデータ・ファイル内での出現順と同じになることを保証できません。 ID 列値の割り当てはロード・ユーティリティーが並列で管理するため、これらの値は任意の順に割り当てられます。これについての例外は、以下のとおりです。

- 単一パーティション・データベースでは、CPU_PARALLELISM が 1 に設定されているとき、行は並列に処理されません。この場合、ID 列値は暗黙のうちに、データ・ファイル・パラメーター内での行の出現順と同じ順序で割り当てられます。
- 複数パーティション・データベースでは、ID 列値が分散キーにある場合、および単一パーティション・エージェントがある場合 (つまり、複数パーティション・エージェントまたは anyorder ファイル・タイプ修飾子を指定しない場合)、ID 列値はデータ・ファイルでの行の出現順と同じ順序で割り当てられます。

表をパーティション・データベースにロードするときに、表のパーティション・キーに ID 列があり、identityoverride 修飾子が指定されていない場合、SAVECOUNT オプションを指定できません。パーティション・キーに ID 列があり、ID 値が生成

されている場合、少なくとも 1 つのデータベース・パーティション上のロード・フェーズからロードを再始動するには、ロード・フェーズの最初からロード全体を再始動する必要があります。これは、整合点が見つからないことを意味します。

注: 以下の基準すべてが満たされている場合、ロード RESTART 操作は許可されません。

- ロードされている表がパーティション・データベース環境にあり、分散キーにある ID 列、または分散キーの一部となる生成列で参照される ID 列のいずれかを少なくとも 1 つその表に含む。
- identityoverride 修飾子が指定されていない。
- 失敗した前回のロード操作には、ロード・フェーズの後に失敗したデータベース・パーティションのロードが含まれていた。

代わりに LOAD TERMINATE または LOAD REPLACE 操作を実行してください。

ID 列が含まれている表にデータをロードする操作を簡略化するには、ファイル・タイプ修飾子として、identitymissing、 identityignore、および identityoverride の 3 つのうちのいずれかを使用する方法があります。これらの指定は相互に排他的です。

ID 列を持たないデータのロード

identitymissing 修飾子は、入力データ・ファイルに ID 列の値が入っていない (NULL すらない) 場合に、 ID 列を持つ表のロードを容易にします。例えば、次のような SQL ステートメントで定義された表があるとします。

```
create table table1 (c1 varchar(30),
                     c2 int generated by default as identity,
                     c3 decimal(7,2),
                     c4 char(1))
```

ID 列を持たない表からエクスポートされたファイル (load.del) からデータを TABLE1 にロードする場合、以下の例を参照してください。

```
Robert, 45.2, J
Mike, 76.9, K
Leo, 23.4, I
```

このファイルをロードする 1 つの方法は、次のように **LOAD** コマンドを使用して、ロードする列を明示的にリストすることです。

```
db2 load from load.del of del replace into table1 (c1, c3, c4)
```

ただし、多くの列を持つ表の場合、この構文は扱いにくく、誤りを生じる可能性があります。ファイルをロードする別の方法は、次のように identitymissing ファイル・タイプ修飾子を使うことです。

```
db2 load from load.del of del modified by identitymissing
   replace into table1
```

このコマンドを実行すると、データ・ファイルの 3 つの列が TABLE1 の c1、c3、および c4 にロードされます。値は c2 の各行に生成されます。

ID 列を持つデータのロード

identityignore 修飾子を指定すると、ロード・ユーティリティーは、入力データ・ファイルに ID 列用のデータが入っていても、そのデータを無視して、各行ごとに ID 値を生成します。例えば、上記で定義したように、ユーザーが次のようなデータの入ったデータ・ファイル (load.del) から TABLE1 をロードするとします。

```
Robert, 1, 45.2, J
Mike, 2, 76.9, K
Leo, 3, 23.4, I
```

ユーザー指定値の 1、2、および 3 が ID 列に使われない場合は、次のような **LOAD** コマンドを発行することができます。

```
db2 load from load.del of del method P(1, 3, 4)
   replace into table1 (c1, c3, c4)
```

ここでも、表に多くの列があると、このアプローチは扱いにくく、誤りを生じる可能性があります。次のように identityignore 修飾子を使用すると、構文が単純化されます。

```
db2 load from load.del of del modified by identityignore
   replace into table1
```

ユーザー指定値を持つデータのロード

identityoverride 修飾子は、 GENERATED ALWAYS ID 列をもつ表にユーザー指定値をロードするために使用します。これが非常に役立つのは、別のデータベース・システムからデータをマイグレーションするときに GENERATED ALWAYS として表を定義しなければならない場合、または **ROLLFORWARD DATABASE** コマンドで DROPPED TABLE RECOVERY オプションを使用してリカバリーしたデータから表をロードする場合です。この修飾子を使用した場合、 ID 列でデータ (または NULL データ) の入っていない行はリジェクトされます (SQL3116W)。この修飾子を使用すると、 GENERATED ALWAYS 列の固有性特性に違反する可能性があることも覚えておいてください。この違反が発生した場合にはまず LOAD TERMINATE 操作を実行し、その後、LOAD INSERT または LOAD REPLACE 操作を実行してください。

生成列のロードに関する考慮事項

入力データに生成列の値があるかどうかに関係なく、ID 列以外の生成列が含まれている表にデータをロードできます。ロード・ユーティリティーは列値を生成します。

生成列関連のファイル・タイプ修飾子が使用されない場合、ロード・ユーティリティーは次のような規則に従って動作します。

- データ・ファイルの対応する行に列の値が欠落している場合や NULL 値が提供されている場合には、生成列に値が作成されます。生成列に非 NULL 値を指定すると、その行はリジェクトされます (SQL3550W)。
- NULL 可能列でない生成列用に NULL 値が作成されると、データの行全体がリジェクトされます (SQL0407N)。これが起きるのは、例えば、NULL 可能列でない生成列が 2 つの表の列の合計として定義されていて、それらの表の列にデータ・ファイルに NULL 値が組み込まれた場合です。

生成列が含まれている表にデータをロードする操作を簡略化するには、ファイル・タイプ修飾子として、generatedmissing、 generatedignore、および generatedoverride の 3 つのうちのいずれかを使用する方法があります。これらの指定は相互に排他的です。

生成列のないデータのロード

generatedmissing 修飾子は、表内に存在する生成列の値が入力データ・ファイル内に入っていない (NULL すらない) 場合に、生成列を持つ表のロードを容易にします。例えば、次のような SQL ステートメントで定義された表があるとします。

```
CREATE TABLE table1 (c1 INT,
                     c2 INT,
                     g1 INT GENERATED ALWAYS AS (c1 + c2),
                     g2 INT GENERATED ALWAYS AS (2 * c1),
                     c3 CHAR(1))
```

生成列を持たない表からエクスポートされたファイル (load.del) からデータを TABLE1 にロードする場合、以下の例を参照してください。

```
1, 5, J
2, 6, K
3, 7, I
```

このファイルをロードする 1 つの方法は、次のように LOAD コマンドを使用して、ロードする列を明示的にリストすることです。

```
DB2 LOAD FROM load.del of del REPLACE INTO table1 (c1, c2, c3)
```

ただし、多くの列を持つ表の場合、この構文は扱いにくく、誤りを生じる可能性があります。ファイルをロードする別の方法は、次のように generatedmissing ファイル・タイプ修飾子を使うことです。

```
DB2 LOAD FROM load.del of del MODIFIED BY generatedmissing
   REPLACE INTO table1
```

このコマンドを実行すると、データ・ファイルの 3 つの列が TABLE1 の c1、c2、および c3 にロードされます。 generatedmissing 修飾子により、TABLE1 の列 g1 と g2 の値が自動的に生成されます。それらの値はデータ・ファイル列にマップされません。

生成列のあるデータのロード

generatedignore 修飾子を指定すると、ロード・ユーティリティーは、ターゲット表にあるすべての生成列用のデータが入力データ・ファイルに入っていても、そのデータを無視して、生成される各列に計算された値をロードします。例えば、上記で定義したように、次のようなデータの入ったデータ・ファイル (load.del) から TABLE1 をロードするとします。

```
1, 5, 10, 15, J
2, 6, 11, 16, K
3, 7, 12, 17, I
```

生成列に関連したファイル・タイプ修飾子が使用されない場合、ユーザー指定の非 NULL 値 10、11、12 (g1 の場合)、および 15、16、17 (g2 の場合) により、行はリジェクトされます (SQL3550W)。これを回避するには、次のような LOAD コマンドを発行します。

```
DB2 LOAD FROM load.del of del method P(1, 2, 5)
   REPLACE INTO table1 (c1, c2, c3)
```

ここでも、表に多くの列があると、このアプローチは扱いにくく、誤りを生じる可能性があります。次のように generatedignore 修飾子を使用すると、構文が単純化されます。

```
DB2 LOAD FROM load.del of del MODIFIED BY generatedignore
   REPLACE INTO table1
```

このコマンドを実行すると、データ・ファイルの列が TABLE1 の c1 (データ 1、2、3 が入る)、c2 (データ 5、6、7 が入る)、および c3 (データ J、K、I が入る) にロードされます。 generatedignore 修飾子により TABLE1 の列 g1 と g2 の値が自動的に生成され、データ・ファイルの列 (10、11、12 および 15、16、17) は無視されます。

ユーザー指定値を持つデータのロード

generatedoverride 修飾子は、生成列をもつ表にユーザー指定値をロードするために使用します。これが役立つのは、別のデータベース・システムからデータをマイグレーションする場合、または **ROLLFORWARD DATABASE** コマンドの RECOVER DROPPED TABLE オプションを使用してリカバリーしたデータから表をロードする場合です。この修飾子を使用した場合、 NULL 不能の生成列でデータ (または NULL データ) の入っていない行はリジェクトされます (SQL3116W)。

この修飾子が使用される場合、ロード操作の後、表は SET INTEGRITY ペンディング状態になります。ユーザー指定値を検証せずに、表を SET INTEGRITY ペンディング状態から解放するには、以下のコマンドを発行します。

```
SET INTEGRITY FOR table-name GENERATED COLUMN IMMEDIATE
  UNCHECKED
```

表を SET INTEGRITY ペンディング状態から解放し、ユーザー指定値を強制的に検査するには、以下のコマンドを発行します。

```
SET INTEGRITY FOR table-name IMMEDIATE CHECKED
```

生成される列がパーティション・キー、ディメンション・キー、または分散キーのいずれかにある場合、 generatedoverride 修飾子は無視され、ロード・ユーティリティーは generatedignore 修飾子が指定された場合のように値を生成します。これは、ユーザー指定の生成列値がその生成列の定義と矛盾するようなシナリオを回避するために行われます。そのような矛盾があると、結果として生成されるレコードは不適切な物理ロケーション (例えば不適切なデータ・パーティション、MDC ブロック、またはデータベース・パーティション) に配置されてしまいます。

注: 列値の生成がロードでサポートされないケースが 1 つあります。生成された列式のうちの 1 つに、FENCED であるユーザー定義関数が入っている場合です。そのような表へのロードを試みると、ロード操作は失敗します。ただし、generatedoverride ファイル・タイプ修飾子を使えば、その種の生成列に自分独自の値を指定することができます。

CURSOR ファイル・タイプを使用したデータの移動

LOAD コマンドを使用する際に CURSOR ファイルを指定することにより、中間エクスポート・ファイルを作成しなくても、 SQL 照会の結果を直接ターゲット表にロードすることができます。

さらに、SQL 照会内でニックネームを参照するか、DECLARE CURSOR ステートメント内で DATABASE オプションを使用するか、または API インターフェースの使用時に sqlu_remotefetch_entry メディア項目を使用することによって、別のデータベースからデータをロードすることができます。

CURSOR ファイル・タイプを使ってデータを移動するための方法は 3 つあります。 1 つ目の方法はコマンド行プロセッサー (CLP) を使用する方法で、2 つ目は API を使用する方法、3 つ目は ADMIN_CMD プロシージャーを使用する方法です。 CLP と ADMIN_CMD プロシージャーの主な違いが以下の表で略述されています。

表 15. CLP と ADMIN_CMD プロシージャーの違い。

相違	**CLP**	**ADMIN_CMD プロシージャー**
構文	照会ステートメント、およびカーソルで使用されるソース・データベースは、LOAD コマンドの外部で、DECLARE CURSOR ステートメントを使って定義されます。	照会ステートメント、およびカーソルで使用されるソース・データベースは、LOAD コマンドの内部で、LOAD from (DATABASE database-alias query-statement) を使って定義されます。

表 15. *CLP と ADMIN_CMD プロシージャーの違い。 (続き)*

相違	CLP	ADMIN_CMD プロシージャー
別のデータベースにアクセスするためのユーザー許可	現在接続しているものとは別のデータベースにデータがある場合、DATABASE キーワードを DECLARE CURSOR ステートメント内で使用する必要があります。同ステートメントにユーザー ID とパスワードを指定することもできます。 DECLARE CURSOR ステートメントにユーザー ID およびパスワードを指定しない場合、ソース・データベースの接続に対して明示的に指定するユーザー ID およびパスワードがターゲット・データベースのアクセスに使用されます。	現在接続しているものとは別のデータベースにデータがある場合、照会ステートメントの前に LOAD コマンドで DATABASE キーワードを使用する必要があります。ターゲット・データベースにアクセスするには、ソース・データベースの接続に対して明示的に指定したユーザー ID とパスワードが必要です。ソース・データベースにユーザー ID またはパスワードを指定することはできません。そのため、ターゲット・データベースとの接続確立時にユーザー ID およびパスワードを指定しなかった場合、または指定されたユーザー ID とパスワードを使ってソース・データベースに対して認証を行えない場合、ADMIN_CMD プロシージャーを使ってロードを実行することはできません。

CLP から LOAD FROM CURSOR 操作を実行するには、まず SQL 照会に対してカーソルを宣言しなければなりません。これが宣言できたら、宣言されるカーソルの名前を *cursorname* に、また CURSOR をファイル・タイプにして、**LOAD** コマンドを発行できます。

以下に例を示します。

1. 以下の定義に従い、ソース表とターゲット表の両方が同じデータベースに存在すると仮定します。

   表 ABC.TABLE1 には次の 3 つの列があります。

   - ONE INT
   - TWO CHAR(10)
   - THREE DATE

   表 ABC.TABLE2 には次の 3 つの列があります。

   - ONE VARCHAR
   - TWO INT
   - THREE DATE

以下の CLP コマンドを実行すると、すべてのデータが ABC.TABLE1 から ABC.TABLE2 にロードされます。

```
DECLARE mycurs CURSOR FOR SELECT TWO, ONE, THREE FROM abc.table1
   LOAD FROM mycurs OF cursor INSERT INTO abc.table2
```

注: 上記の例では、CLP を介して SQL 照会からロードする方法を示します。ただし、db2Load API を介して SQL 照会からロードすることもできます。*sqlu_statement_entry* 構造および SQLU_SQL_STMT メディア・タイプを使用するには sqlu_media_list 構造の *piSourceList* を定義します。そして *piFileType* の値を SQL_CURSOR として定義します。

2. 以下の定義に従い、ソース表とターゲット表がそれぞれ異なるデータベースに存在すると仮定します。

データベース「dbsource」の表 ABC.TABLE1 には次の 3 つの列があります。

- ONE INT
- TWO CHAR(10)
- THREE DATE

データベース「dbtarget」の表 ABC.TABLE2 には次の 3 つの列があります。

- ONE VARCHAR
- TWO INT
- THREE DATE

フェデレーションを可能にし、データ・ソース ('dsdbsource') をカタログした場合は、次の例に示すとおり、ソース・データベースに対してニックネームを宣言した後、このニックネームに対してカーソルを宣言し、FROM CURSOR オプションを指定して LOAD コマンドを呼び出すことができます。

```
CREATE NICKNAME myschema1.table1 FOR dsdbsource.abc.table1
DECLARE mycurs CURSOR FOR SELECT TWO,ONE,THREE FROM myschema1.table1
LOAD FROM mycurs OF cursor INSERT INTO abc.table2
```

あるいは以下の例で示されているように、DECLARE CURSOR ステートメントの DATABASE オプションを使用することもできます。

```
DECLARE mycurs CURSOR DATABASE dbsource USER dsciaraf USING mypasswd
FOR SELECT TWO,ONE,THREE FROM abc.table1
LOAD FROM mycurs OF cursor INSERT INTO abc.table2
```

DECLARE CURSOR ステートメントの DATABASE オプション (ロード API 使用時の remotefetch メディア・タイプに相当する) を使用することには、ニックネームのアプローチに勝る利点があります。

パフォーマンス

remotefetch メディア・タイプを使ったデータのフェッチは、ロード操作と緊密に統合されています。ニックネームのアプローチと比べて、レコード・フェッチのための遷移の層が少なくなります。さらに、複数パーティション・データベースでソース表とターゲット表が全く均等に分散されている場合、ロード・ユーティリティーはデータのフェッチを並列的に処理します。これにより、パフォーマンスは向上します。

使いやすさ

フェデレーションの使用可能化、リモート・データ・ソースの定義、またはニックネームの宣言は不要です。必要なのは DATABASE オプション (および必要な場合は USER および USING オプション) の指定だけです。

このメソッドはカタログされているデータベースで使用できますが、ニックネームの使用は、簡単にカタログできない各種データ・ソースからのフェッチのための堅固な機構を提供します。

この remotefetch 機能をサポートするために、ロード・ユーティリティーは SOURCEUSEREXIT 機能をサポートするインフラストラクチャーを利用します。ロード・ユーティリティーは、アプリケーションとして実行されるプロセスを作成し、それによってソース・データベースへの接続を管理し、フェッチを実行します。このアプリケーションはロード・ユーティリティーが実行されるトランザクションではなく、固有のトランザクションと関連付けられます。

注:

1. 上記の例では、CLP を介して DECLARE CURSOR ステートメントの DATABASE オプションを使用することにより、カタログされているデータベースに対する SQL 照会からロードを行う方法を示しています。しかし、db2Load API を介して、カタログされているデータベースに対する SQL 照会からロードを行うこともできます。その場合は、*db2LoadStruct* 構造の *piSourceList* および *piFileTypevalues* を、それぞれ sqlu_remotefetch_entry メディア項目および SQLU_REMOTEFETCH メディア・タイプを使用するよう定義します。
2. 上記の例で示したとおり、SQL 照会のソース列タイプはターゲット列タイプと互換性がなければなりませんが、同一である必要はありません。

制約事項

DATABASE オプションを使用して定義されているカーソルからロードする場合 (db2Load API で sqlu_remotefetch_entry メディア項目を使用する場合も同様)、次の制約事項が適用されます。

1. SOURCEUSEREXIT オプションを並行して指定することはできません。
2. METHOD N オプションはサポートされません。
3. usedefaults ファイル・タイプ修飾子はサポートされません。

従属即時ステージング表の伝搬

ロードされる表が即時伝搬属性を持つステージング表の基礎表であり、ロード操作が挿入モードで実行される場合には、従属即時ステージング表への後続の伝搬は増分になります。

増分伝搬時には、基礎表にある追加行に対応する行がステージング表に追加されます。基礎表が大きく、追加データが少ない場合には、増分伝搬の速度はそれだけ速くなります。また、ステージング表を使用して、その従属据え置きマテリアライズ照会表をリフレッシュすると、パフォーマンスが改善されます。増分伝搬が許可されず、ステージング表に不完全というマークが付けられる場合もあります。つまり、CONST_CHECKED 列のステージング・バイトの値は F になります。この状態では、ステージング表を使用して、従属据え置きマテリアライズ照会表をリフレッシュすることはできず、マテリアライズ照会表の保守プロセスでフル・リフレッシュが必要になります。

表が不完全状態で、INCREMENTAL オプションが指定されているにもかかわらず、表の増分伝搬が実行できない場合、エラーが戻されます。以下のいずれかが発生すると、システムは即時データ伝搬をオフにし、表状態を不完全に設定します。

- ステージング表の基礎表でロード置換操作が実行されるか、または基礎表に対する最後の整合性チェックの後で NOT LOGGED INITIALLY WITH EMPTY TABLE オプションが活動化された場合。
- ステージング表の従属マテリアライズ照会表、またはステージング表が REPLACE または INSERT モードでロードされた場合。
- 整合性チェック時に FULL ACCESS オプションを使用することによってステージング表が伝搬される前に、基礎表が SET INTEGRITY ペンディング状態ではなくなった場合。
- ステージング表の基礎表で、整合性のチェックが非増分的に実行された場合。
- ステージング表またはその基礎表が入った表スペースがある時点でロールフォワードされており、ステージング表とその基礎表が異なる表スペースに常駐する場合。

ステージング表で、 SYSCAT.TABLES カタログの CONST_CHECKED 列に W 値があり、 NOT INCREMENTAL オプションが指定されていない場合、ステージング表への増分伝搬が実行され、 SYSCAT.TABLES の CONST_CHECKED 列には、すべてのデータをシステムがチェックしたわけではないことを示す U というマークが付けられます。

以下の例は、ステージング表 G1 の基礎表 UT1 およびその従属据え置きマテリアライズ照会表 AST1 へのロード挿入操作を示します。このシナリオでは、UT1 の整合性チェックと AST1 のリフレッシュの両方が増分的に処理されます。

```
LOAD FROM IMTFILE1.IXF of IXF INSERT INTO UT1;
LOAD FROM IMTFILE2.IXF of IXF INSERT INTO UT1;
SET INTEGRITY FOR UT1,G1 IMMEDIATE CHECKED;

REFRESH TABLE AST1 INCREMENTAL;
```

従属即時マテリアライズ照会表のリフレッシュ

即時リフレッシュ・マテリアライズ照会表の基礎表が INSERT オプションを使用してロードされる場合には、 REFRESH IMMEDIATE で定義される従属マテリアライズ照会表で SET INTEGRITY ステートメントを実行すると、マテリアライズ照会表で増分リフレッシュが実行されます。

増分リフレッシュ時には、基礎表にある追加行に対応する行が更新され、マテリアライズ照会表に挿入されます。基礎表が大きく、追加データが少ない場合には、増分リフレッシュの速度はそれだけ速くなります。増分リフレッシュが許可されず、フル・リフレッシュ (つまり、マテリアライズ照会表定義照会の再計算) が使用される場合もあります。

INCREMENTAL オプションを指定したにもかかわらず、マテリアライズ照会表の増分的な処理を実行できない場合、以下に示す条件が成立するならエラーが戻されます。

- マテリアライズ照会表の基礎表でロード置換操作が実行されるか、または基礎表に対する最後の整合性チェックの後で NOT LOGGED INITIALLY WITH EMPTY TABLE オプションが活動化された場合。

- マテリアライズ照会表がロードされている場合 (REPLACE または INSERT モードのいずれかで)。
- 整合性チェック時に FULL ACCESS オプションを使用することによってマテリアライズ照会表がリフレッシュされる前に、基礎表が SET INTEGRITY ペンディング状態ではなくなった場合。
- マテリアライズ照会表の基礎表で、整合性のチェックが非増分的に実行された場合。
- マテリアライズ照会表が、アップグレード前の SET INTEGRITY ペンディング状態にあった場合。
- マテリアライズ照会表またはその基礎表が入った表スペースがある時点でロールフォワードされており、マテリアライズ照会表とその基礎表が異なる表スペースに常駐する場合。

マテリアライズ照会表で、SYSCAT.TABLES カタログの CONST_CHECKED 列に 1 つまたは複数の W 値がある場合で、 SET INTEGRITY ステートメントに NOT INCREMENTAL オプションが指定されていない場合、表は増分にリフレッシュされ、SYSCAT.TABLES の CONST_CHECKED 列には、すべてのデータをシステムがチェックしたわけではないことを示す U というマークが付けられます。

以下の例は、マテリアライズ照会表 AST1 の基礎表 UTI へのロード挿入操作を示します。 UT1 でデータ整合性がチェックされ、データ移動不可モードになります。 AST1 の増分リフレッシュが完了すると、UT1 はフル・アクセス状態に戻ります。このシナリオでは、UT1 の整合性チェックと AST1 のリフレッシュの両方が増分的に処理されます。

```
LOAD FROM IMTFILE1.IXF of IXF INSERT INTO UT1;
LOAD FROM IMTFILE2.IXF of IXF INSERT INTO UT1;
SET INTEGRITY FOR UT1 IMMEDIATE CHECKED;
REFRESH TABLE AST1;
```

の MDC および ITC の考慮事項

以下の制約事項はマルチディメンション・クラスタリング (MDC) 表および挿入時クラスタリング (ITC) 表へのデータのロード時に適用されます。

- **LOAD** コマンドの SAVECOUNT オプションはサポートされていません。
- これらの表は独自のフリー・スペースを管理するため、 totalfreespace ファイル・タイプ修飾子はサポートされていません。
- MDC または ITC 表には anyorder ファイル・タイプ修飾子が必要です。 anyorder 修飾子を使用せずに MDC 表または ITC 表へのロードを実行すると、この修飾子はユーティリティーによって明示的に使用可能にされます。

MDC または ITC 表に **LOAD** コマンドを使用する場合には、ユニーク制約の違反は以下のように処理されます。

- ロードするデータと同じユニーク・キーを持つレコードが (ロード操作の開始前に既に) 表に存在する場合、元のレコードは残り、新規レコードが削除フェーズで削除されます。
- ロードするデータと同じユニーク・キーを持つレコードが (ロード操作の開始前には) 表に存在しない場合、ユニーク・キーとそれと重複する (同じユニーク・キーを持つ) レコードの両方が表にロードされる場合には、レコードのうち 1 つだけがロードされ、その他のレコードは削除フェーズで除去されます。

注: どのレコードがロードされて、どのレコードが削除されるかを判別するための明示的な技法はありません。

パフォーマンスの考慮

複数のディメンションがある MDC 表をロードする場合のロード・ユーティリティーのパフォーマンスを改善するには、 *util_heap_sz* データベース構成パラメーターの値を大きくしなければなりません。ユーティリティーで使用できるメモリーを増やすと、 mdc-load アルゴリズムのパフォーマンスが大きく向上します。こうすると、ロード・フェーズで実行されるデータのクラスタリング時に、ディスクの入出力を減らすことができます。バージョン 9.5 から、**LOAD** コマンドの DATA BUFFER オプションの値は、システムにさらに使用可能メモリーがある場合に、*util_heap_sz* を一時的に超えられるようになりました。.

すべての MDC および ITC 表にはブロック索引があるため、MDC または ITC ロード操作には常に構築フェーズがあります。

ロード・フェーズでは、ブロック・マップの保守のために余分のロギングが実行されます。割り振られるエクステントごとに、おおよそ 2 つの余分のログ・レコードがあります。パフォーマンスを良くするためには、このことを考慮に入れた値に *logbufsz* データベース構成パラメーターを設定する必要があります。

MDC および ITC 表にデータをロードするために、索引付きのシステム一時表が使用されます。表のサイズはロードされる個々のセルの数に比例します。表にあるそれぞれの行のサイズは MDC 次元キーのサイズに比例します。 ITC 表には 1 つのセルしかなく、2 バイトのディメンション・キーを使用します。ロード操作時にこの表の操作によるディスク入出力を最小限に抑えるには、 TEMPORARY 表スペースのバッファー・プールの大きさが十分であることを確認してください。

カスタマイズしたアプリケーション (ユーザー出口) を使用したデータの移動

ロード SOURCEUSEREXIT オプションを使用すると、カスタマイズしたスクリプトまたは実行ファイル (ここではユーザー*出口*と呼びます) をロード・ユーティリティーが実行するための機構が提供されます。

ユーザー出口の目的は、1 つ以上の Named PIPE に、ロード・ユーティリティーによって同時に読み取られるデータを取り込むことです。複数パーティション・データベースでは、ユーザー出口の複数のインスタンスを同時に呼び出すことにより、入力データの並列処理を実現することができます。

72 ページの図 5 で示されているとおり、ロード・ユーティリティーは 1 つ以上の Named PIPE を作成し、カスタマイズされた実行ファイルを実行するためのプロセスを作成します。ユーザー出口による Named PIPE へのデータのフィードとロード・ユーティリティーによる読み取りは並行して行われます。

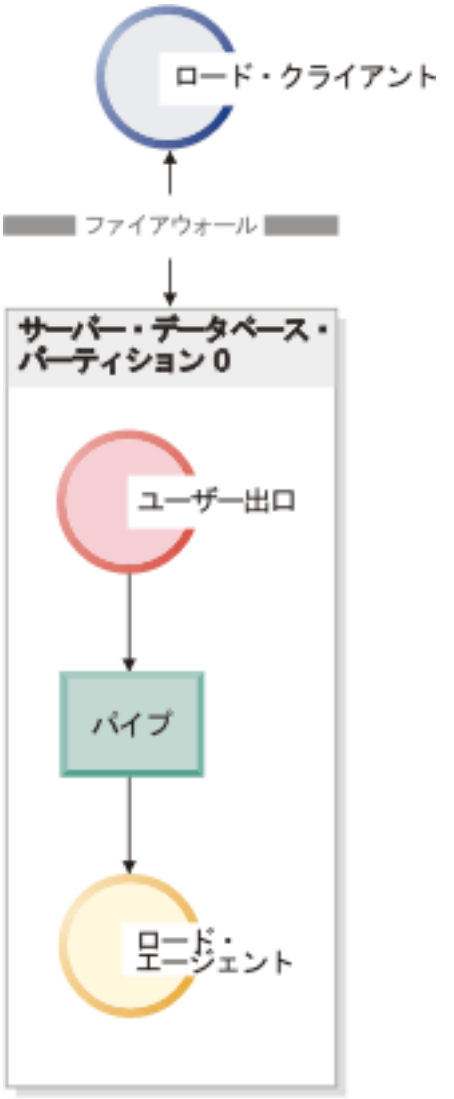


図 5. ロード・ユーティリティーは着信データをパイプから読み取って、処理します。

パイプにフィードされるデータは、指定されるロード・オプション (ファイル・タイプおよびファイル・タイプ修飾子を含む) を反映しなければなりません。ロード・ユーティリティーは、指定されるデータ・ファイルを直接的には読み取りません。代わりに、指定されるデータ・ファイルはユーザー出口に対する引数として、その実行時に渡されます。

ユーザー出口の呼び出し

ユーザー出口は DB2 インストール・ディレクトリー (多くの場合 sqllib) の bin サブディレクトリーになければなりません。ロード・ユーティリティーは、次のコマンド行引数を使ってユーザー出口の実行ファイルを呼び出します。

```
<base pipename> <number of source media>
<source media 1> <source media 2> ... <user exit ID>
<number of user exits> <database partition number>
```

各部分の定義は次のとおりです。

<base pipename >
ロード・ユーティリティーによって作成され、データの読み取り元となる Named PIPE のベース名です。ユーティリティーは、LOAD コマンドに対して提供される各ソース・ファイルにつき 1 つのパイプを作成します。これらのパイプには .xxx が付加されます。xxx は、提供されるソース・ファイルの索引です。例えば、LOAD コマンドに 2 つのソース・ファイルが提供され、ユーザー出口に渡される <base pipename> 引数が pipe123 である場合、ユーザー出口がデータをフィードする 2 つの名前付きパイプは pipe123.000 および pipe123.001 となります。パーティション・データベース環境では、ロード・ユーティリティーは基本のパイプ名にデータベース・パーティション (DBPARTITION) 番号 .yyy を付加して、pipe123.xxx.yyy. というパイプ名になります。

<number of source media>
後に続くメディア引数の数です。

<source media 1> <source media 2> ...
LOAD コマンドで指定される 1 つ以上のソース・ファイルのリストです。各ソース・ファイルは二重引用符で囲まれます。

<user exit ID>
PARALLELIZE オプションを使用可能にした場合に使うことができる特殊値です。この整数値 (1 から N の範囲で、N は作成されるユーザー出口の総数) は、実行中のユーザー出口の特定のインスタンスを識別します。PARALLELIZE オプションが使用可能でない場合、この値はデフォルトの 1 になります。

<number of user exits>
PARALLELIZE オプションを使用可能にした場合に使うことができる特殊値です。この値は、現在実行中のユーザー出口の総数を表します。 PARALLELIZE オプションが使用可能でない場合、この値はデフォルトの 1 になります。

<database partition number>
PARALLELIZE オプションを使用可能にした場合に使うことができる特殊値です。これはユーザー出口が実行しているデータベース・パーティション (DBPARTITION) 番号です。 PARALLELIZE オプションが使用可能でない場合、この値はデフォルトの 0 になります。

追加のオプションとフィーチャー

以下のセクションでは、SOURCEUSEREXIT 機能の追加オプションについて説明します。

REDIRECT
このオプションにより、ユーザー出口プロセスの STDIN ハンドルにデータを渡すか、または STDOUT および STDERR ハンドルからデータを取り込むことが可能になります。

INPUT FROM BUFFER <buffer>
これにより、ユーザー出口の STDIN 入力ストリームに直接情報を渡すことが可能になります。ユーザー出口を実行するプロセスを作成した後、ロード・ユーティリティーはこの新規プロセスの STDIN のファイル記述子を獲得し、指定されたバッファーに渡します。ユーザー出口は STDIN を読み取

って情報を獲得します。ロード・ユーティリティーは STDIN を使用して <buffer> の内容をユーザー出口に送るだけで、その内容の解釈や変更は行いません。例えば、ユーザー出口が STDIN から 8 バイトのユーザー ID と 8 バイトのパスワードの 2 つの値を読み取るように設計されている場合、C で作成されたユーザー出口実行ファイルには次の行が含まれるかもしれません。

```
rc = read (stdin, pUserID, 8);
rc = read (stdin, pPasswd, 8);
```

ユーザーは、以下の LOAD コマンドに示すように、INPUT FROM BUFFER オプションを使ってこの情報を渡すことができます。

```
LOAD FROM myfile1 OF DEL INSERT INTO table1
SOURCEUSEREXIT myuserexit1 REDIRECT INPUT FROM BUFFER myuseridmypasswd
```

注: ロード・ユーティリティーは <buffer> のサイズを LOB 値の最大サイズに制限します。しかし、コマンド行プロセッサー (CLP) の場合は、<buffer> のサイズは CLP ステートメントの最大サイズに制限されます。また、CLP の場合、<buffer> に含める文字を従来の ASCII 文字のみにすることも勧められています。 db2Load API を使ってロード・ユーティリティーが呼び出される場合、あるいはその代わりに INPUT FROM FILE オプションが使用される場合、これらの問題を回避できます。

INPUT FROM FILE <filename>
クライアント・サイドのファイルの内容をユーザー出口の STDIN 入力ストリームに直接渡すことが可能になります。このオプションはほぼ INPUT FROM BUFFER オプションと同じですが、このオプションには潜在的 CLP の制限がありません。ファイル名はクライアント・サイドの完全修飾ファイルでなければならず、LOB 値の最大サイズ以下でなければなりません。

OUTPUT TO FILE <filename>
ユーザー出口プロセスの STDOUT および STDERR ストリームを取り込み、それをサーバー・サイドのファイルに入れることが可能になります。ユーザー出口実行可能ファイルを実行するプロセスを作成した後、ロード・ユーティリティーはこの新規プロセスの STDOUT および STDERR ハンドルを指定のファイル名にリダイレクトします。このオプションは、ユーザー出口内のエラーおよびアクティビティーのデバッグとロギングを行うのに便利です。ファイル名は、サーバー・サイドの完全修飾ファイルでなければなりません。 PARALLELIZE オプションが有効になっている場合は、ユーザー出口ごとに 1 つのファイルが存在し、それぞれのファイルには 3 桁の数値 ID が付加されます (例えば *filename.000*)。

PARALLELIZE
このオプションは、複数のユーザー出口プロセスを同時に呼び出すことにより、ロード・ユーティリティーに入るデータのスループットを向上させることができます。このオプションは、複数パーティション・データベースにのみ適用できます。ロード操作中にデータが複数のデータベース・パーティションに分散される場合、呼び出されるユーザー出口インスタンスの数はパーティション・エージェントの数と同じになります。ロード操作中にデータが複数のデータベース・パーティションに分散されない場合は、ロードするエージェントの数と同じになります。

各ユーザー出口に渡される <user exit ID>、<number of user exits>、および <database partition number> 引数は、それぞれユーザー出口の固有 ID (1 から N)、総数 (N)、およびユーザー出口インスタンスが実行されているデータベース・パーティション (DBPARTITION) 番号を表します。各ユーザー出口プロセスが Named PIPE に書き出すどのデータも、他の並行プロセスによって複写されることがないようにしなければなりません。ユーザー出口アプリケーションがそうするための方法はたくさんありますが、これらの値はデータの複写を防ぐ上で役に立つかもしれません。例えば、各データのレコードに固有の整数の列値が含まれる場合、ユーザー出口アプリケーションは <user exit ID> および <number of user exits> の値を使用して、各ユーザー出口インスタンスが固有の結果セットを自分の名前付きパイプに戻すようにすることができます。ユーザー出口アプリケーションは、次のように **MODULUS** プロパティーを使用することができるかもしれません。

```
i = <user exit ID>
N = <number of user exits>

foreach record
{
   if ((unique-integer MOD N) == i)
   {
     write this record to my named-pipe
   }
}
```

作成されるユーザー出口プロセスの数は、データベース・パーティションに指定される分散モードによって異なります。

1. 76 ページの図 6 が示すように、PARTITION_AND_LOAD (デフォルト) または PARTITION_ONLY (PARALLEL なし) が指定されている場合は、事前パーティション化エージェントごとに 1 つのユーザー出口プロセスが作成されます。 .

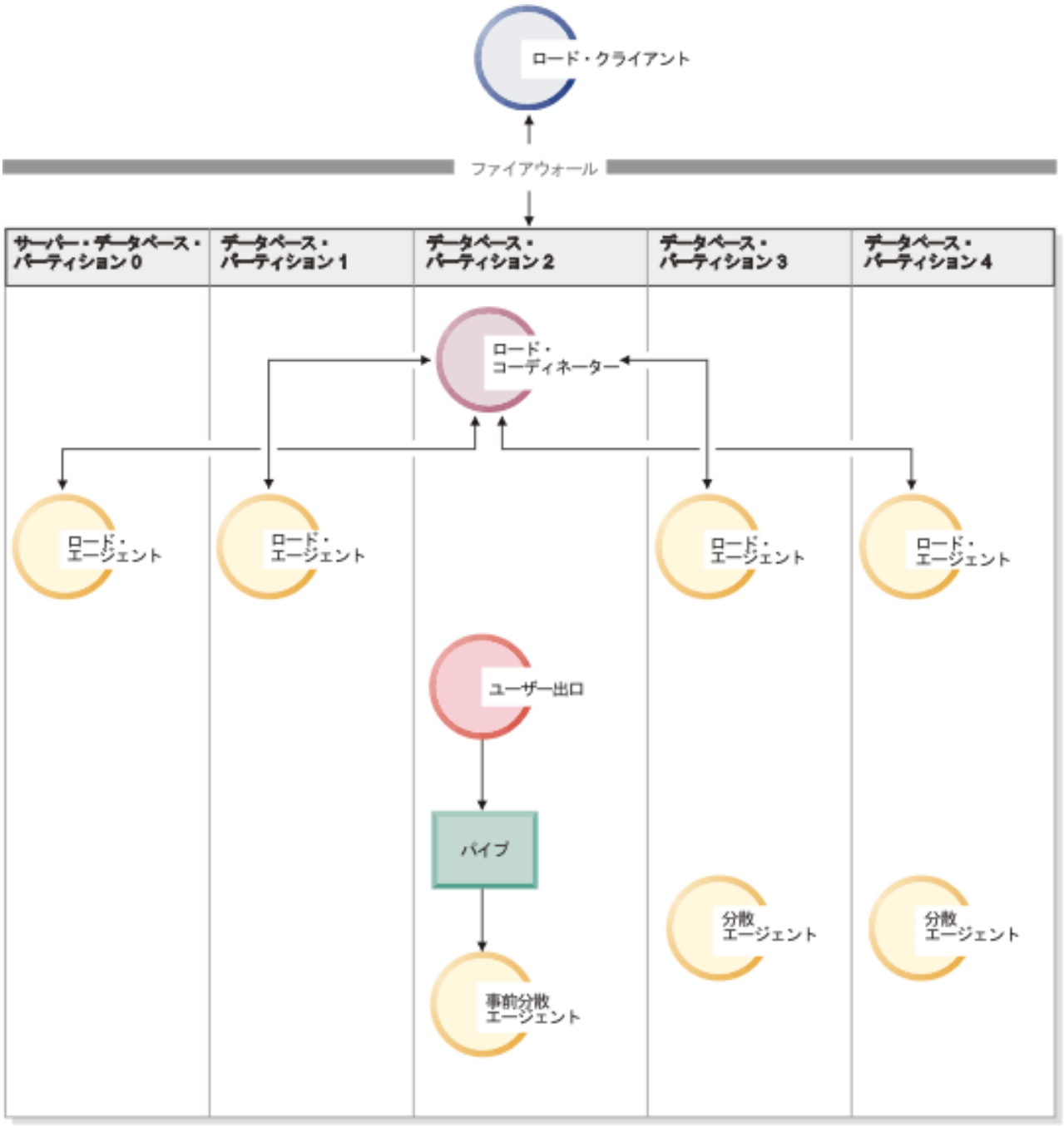


図 6. PARTITION_AND_LOAD (デフォルト) または PARTITION_ONLY (PARALLEL なし) が指定されている場合に実行される各種タスク。

2. 77 ページの図 7 が示すように、PARTITION_AND_LOAD (デフォルト) または PARTITION_ONLY (PARALLEL あり) が指定されている場合は、パーティション化エージェントごとに 1 つのユーザー出口プロセスが作成されます。

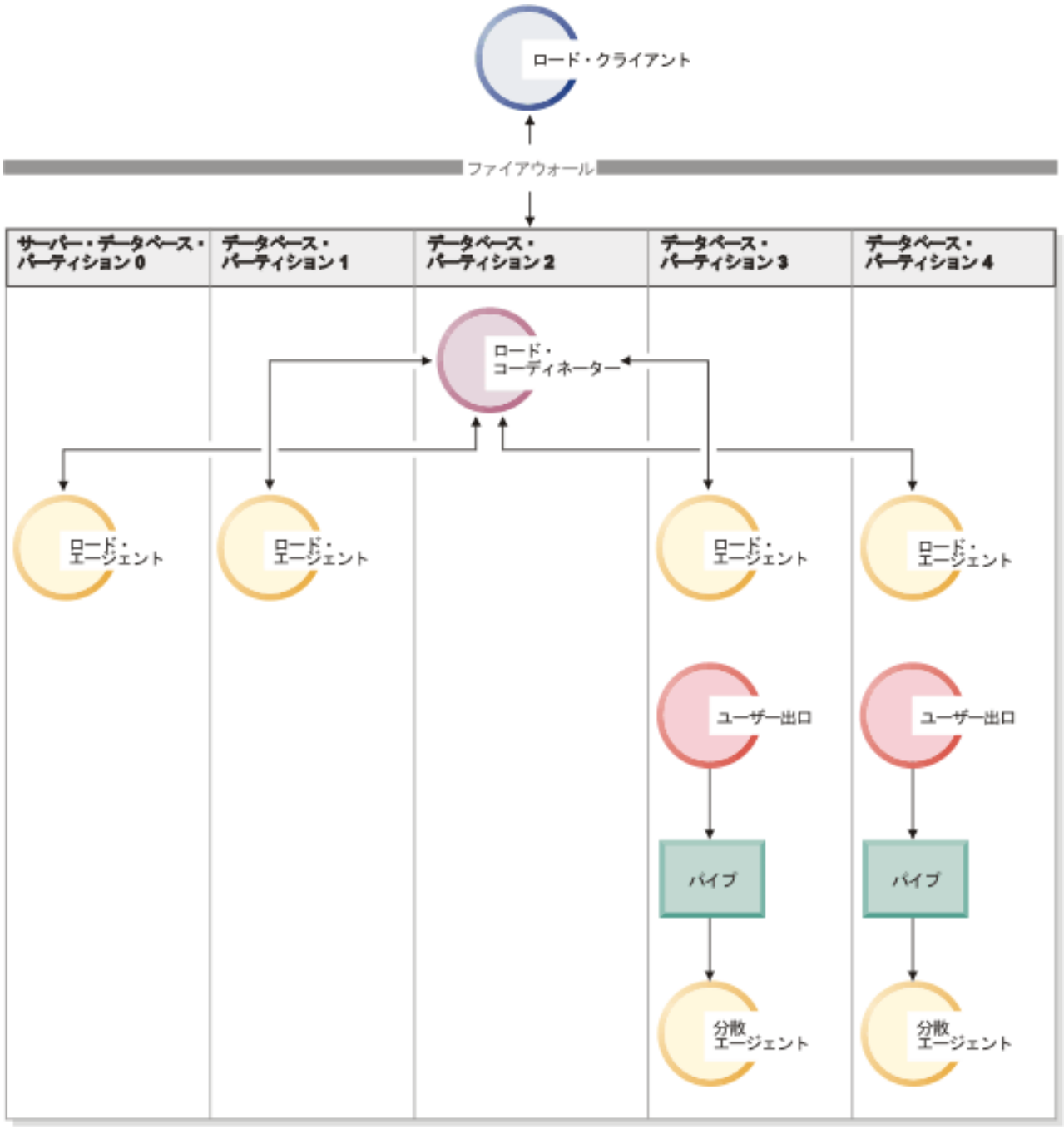


図 7. PARTITION_AND_LOAD (デフォルト) または PARTITION_ONLY (PARALLEL あり) が指定されている場合に実行される各種タスク。

3. 78 ページの図 8 が示すように、 LOAD_ONLY または LOAD_ONLY_VERIFY_PART が指定されている場合は、ロード・エージェントごとに 1 つのユーザー出口プロセスが作成されます。

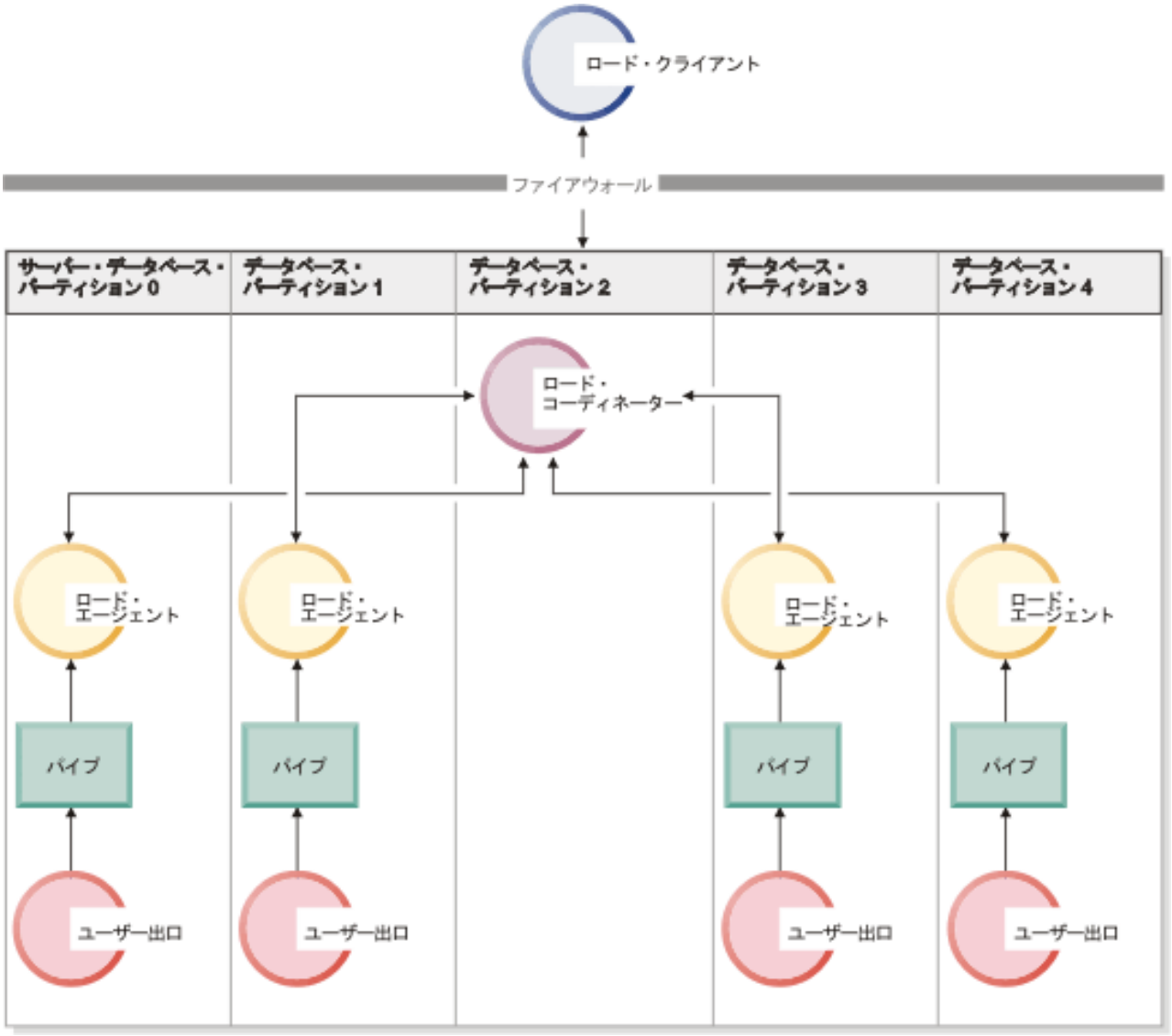


図 8. LOAD_ONLY または LOAD_ONLY_VERIFY_PART が指定される場合に実行される各種タスク。

制約事項

- SOURCEUSEREXIT PARALLELIZE オプションを指定しないと、 LOAD_ONLY および LOAD_ONLY_VERIFY_PART の partitioned-db-cfg モード・オプションはサポートされません。

例

例 1: ソース・メディア・ファイルの各レコードから、すべてのタブ文字 '¥t' をコンマ文字 ',' に置き換えるロード・ユーザー出口スクリプト。このユーザー出口スクリプトを使用してロード・ユーティリティーを起動するには、以下のようなコマンドを使用します。

```
DB2 LOAD FROM /path/file1 OF DEL INSERT INTO schema1.table1
SOURCEUSEREXIT example1.pl REDIRECT OUTPUT TO FILE /path/ue_msgs.txt
```

このユーザー出口は、sqllib/bin/ フォルダー内に入れなければならず、実行許可が必要であることに注意してください。

以下に example1.pl を示します。

```
#!/bin/perl

# Filename: example1.pl
#
# This script is a simple example of a userexit for the Load utility
# SOURCEUSEREXIT feature. This script will replace all tab characters '¥t'
# with comma characters ',' from every record of the source media file.
#
# To invoke the Load utility using this userexit, use a command similar to:
#
#  db2 LOAD FROM /path/file1 OF DEL INSERT INTO schema1.table1
#   SOURCEUSEREXIT example1.pl REDIRECT OUTPUT TO FILE /path/ue_msgs.txt
#
# The userexit must be placed into the sqllib/bin/ folder, and requires
# execute permissions.
#------------------------------------------------------------------
if ($#ARGV < 5)
{
  print "Invalid number of arguments:¥n@ARGV¥n";
  print "Load utility should invoke userexit with 5 arguments (or more):¥n";
  print "<base pipename> <number of source media> ";
  print "<source media 1> <source media 2> ... <user exit ID> ";
  print "<number of user exits> <database partition number> ";
  print "<optional: redirected input> ¥n";
  die;
}

# Open the output fifo file (the Load utility is reading from this pipe)
#-------------------------------------------------------------------------
$basePipeName = $ARGV[0];
$outputPipeName = sprintf("%s.000", $basePipeName);
open(PIPETOLOAD, '>', $outputPipeName) || die "Could not open $outputPipeName";

# Get number of Media Files
#--------------------------
$NumMediaFiles = $ARGV[1];

# Open each media file, read the contents, replace '¥t' with ',', send to Load
#-------------------------------------------------------------------------------
for ($i=0; $i<$NumMediaFiles; $i++)
{
  # Open the media file
  #--------------------
  $mediaFileName = $ARGV[2+$i];
  open(MEDIAFILETOREAD, '<', $mediaFileName) || die "Could not open $mediaFileName";

  # Read each record of data
  #-------------------------
  while ( $line = <MEDIAFILETOREAD> )
  {
    # Replace '¥t' characters with ','
    #---------------------------------
    $line =~ s/¥t/,/g;

    # send this record to Load for processing
    #----------------------------------------
    print PIPETOLOAD $line;
  }
  # Close the media file
  #---------------------
  close MEDIAFILETOREAD;
}

# Close the fifo
```

```
#---------------
close PIPETOLOAD;

exit 0;
```

LIST UTILITIES コマンドを使用したロード操作のモニター

LIST UTILITIES コマンドを使用して、データベースでのロード操作の進行をモニターできます。

手順

LIST UTILITIES コマンドを使用するには、以下のようにします。
LIST UTILITIES コマンドを発行して、**SHOW DETAIL** パラメーターを指定します。

```
list utilities show detail
```

例

以下は、**LIST UTILITIES** コマンドを使用してロード操作のパフォーマンスをモニターした場合の出力例です。

```
ID                                = 10
Type                              = LOAD
Database Name                     = TEST
Member Number                     = 1
Description                       = OFFLINE LOAD DEL AUTOMATIC INDEXING REPLACE
COPY NO BEER   .TABLE1
Start Time                        = 08/16/2011 08:52:53.861841
State                             = Executing
Invocation Type                   = User
Progress Monitoring:
   Phase Number                   = 1
      Description                 = SETUP
      Total Work                  = 0 bytes
      Completed Work              = 0 bytes
      Start Time                  = 08/16/2011 08:52:53.861865

   Phase Number [Current]         = 2
      Description                 = LOAD
      Total Work                  = 49900 rows
      Completed Work              = 25313 rows
      Start Time                  = 08/16/2011 08:52:54.277687

   Phase Number                   = 3
      Description                 = BUILD
      Total Work                  = 2 indexes
      Completed Work              = 0 indexes
      Start Time                  = Not Started
```

ロードに関する追加の考慮事項

並列処理とロード

ロード・ユーティリティーは、複数のプロセッサーや複数の記憶装置が使用されているハードウェア構成 (対称マルチプロセッサー (SMP) 環境など) を利用します。

ロード・ユーティリティーを使って大容量データの並列処理を実行する方法はいくつかあります。その 1 つは複数の記憶装置を使用する方法であり、ロード操作中に入出力の並列処理が可能になります (81 ページの図 9 を参照)。別の方法は SMP 環境における複数のプロセッサーの使用が関係しており、パーティション内の並列

処理が可能になります (図 10 を参照)。これらの両方の方法を併用すれば、データのロード時間をさらに短くすることができます。

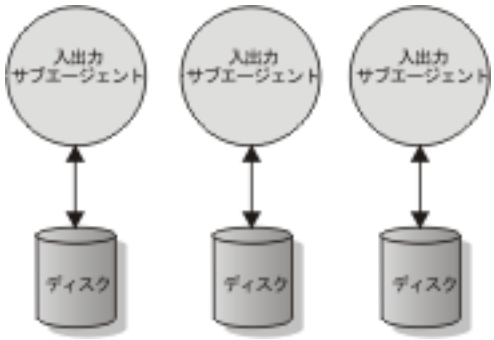


図 9. データ・ロード時に入出力の並列処理を利用する

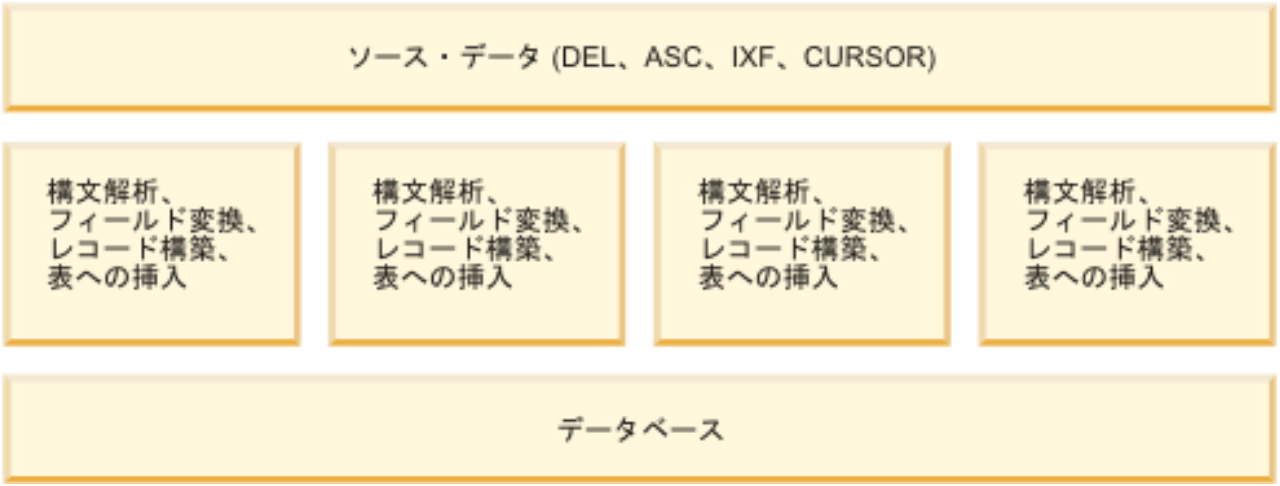


図 10. データ・ロード時のパーティション内の並列処理の利用

ロード操作時の索引作成

索引は、ロード操作の構築フェーズで作成されます。 **LOAD** コマンドで指定できる索引付けモードは 4 つあります。

1. REBUILD。すべての索引を再作成します。
2. INCREMENTAL。索引を新しいデータで拡張します。
3. AUTOSELECT。REBUILD モードと INCREMENTAL モードのどちらにするかはロード・ユーティリティーが自動的に決定します。 AUTOSELECT がデフォルトです。 **LOAD REPLACE** 操作が実行される場合は、REBUILD 索引付けモードが使用されます。ロード REPLACE 操作が実行されない場合、索引付けモードは、新しくロードされたデータの量に対する表内の既存のデータ量の比率に基づいて選択されます。比率が十分に大きい場合は INCREMENTAL 索引付けモードが選択されます。そうでない場合、REBUILD 索引付けモードが選択されます。
4. DEFERRED。このモードが指定されている場合、ロード・ユーティリティーは索引を作成しようとしません。索引には最新表示が必要であるというマークが付けら

れ、最初にアクセスされるときに強制的に再作成されることがあります。次の状況では DEFERRED オプションを使用できません。

- **ALLOW READ ACCESS** オプションが指定される場合 (このオプションは索引を保守せず、索引スキャナーは有効な索引を必要とする)
- 表に対してユニーク索引が定義される場合
- XML データがロードされる場合 (XML Path 索引はユニーク索引であり、デフォルトでは表に XML 列が追加されるたびに作成される)

ALLOW READ ACCESS オプションを指定するロード操作では、選択した索引付けモードによっては、スペース使用量およびロギングに特に配慮する必要があります。**ALLOW READ ACCESS** オプションが指定されると、ロード・ユーティリティーは、索引が再作成中であっても引き続きそれらを照会に使用できるようにします。

ALLOW READ ACCESS モードのロード操作で **INDEXING MODE INCREMENTAL**オプションが指定されると、ロード・ユーティリティーは、索引ツリーの整合性を保護するログ・レコードを書き込みます。書き込まれるログ・レコードの数は、挿入されるキーの数の一部であり、同様の SQL 挿入操作で必要とされる数よりずっと少ない数です。 **ALLOW NO ACCESS** モードのロード操作で **INDEXING MODE INCREMENTAL** オプションが指定されている場合は、通常のスペース割り振りログに加えて小さなログ・レコードのみが書き込まれます。

注: これは、**COPY YES** を指定せず、**logindexrebuild** 構成パラメーターを ON に設定した場合にのみ当てはまります。

ALLOW READ ACCESS モードのロード操作で **INDEXING MODE REBUILD** オプションが指定されると、新しい索引は、元の索引と同じ表スペースまたは SYSTEM TEMPORARY 表スペースのいずれかにシャドー として作成されます。元の索引は変更されずにロード操作で使用することができ、表は排他ロックされたままでロード操作の終わりに新規索引に置き換えられるだけです。ロード操作に失敗してトランザクションがロールバックされる場合でも、元の索引は変更されません。

デフォルトでは、シャドー索引は、元の索引と同じ表スペースに作成されます。元の索引と新規索引の両方が同時に保守されるため、同時に両方の索引を保留できる十分な表スペースがなければなりません。ロード操作が打ち切られると、新規索引の作成に使用される余分のスペースが解放されます。ロード操作がコミットされると、元の索引に使用されるスペースが解放され、新規索引が現行の索引になります。元の索引と同じ表スペースに新規索引が作成されると、元の索引の置換がほとんど同時に行われます。

SMS 表スペースで索引が作成される場合、 .IN1 接尾部および .INX 接尾部のある表スペース・ディレクトリーで索引ファイルを見ることができます。これらの接尾部は、どれが元の索引で、どれがシャドー索引であるかを示しません。ただし、DMS 表スペースで索引が作成される場合、新規のシャドー索引は不可視になります。

索引作成パフォーマンスの改善

SYSTEM TEMPORARY 表スペースでの新規索引の作成

元の表スペースでスペースが不足しないようにするために、新規索引を SYSTEM TEMPORARY 表スペースに作成することができます。 **USE**

tablespace-name オプションを使用すると、**INDEXING MODE REBUILD** オプションおよび **ALLOW READ ACCESS** オプションを使用する際に、索引が SYSTEM TEMPORARY 表スペースで再作成されるようにすることができます。システム一時表は SMS 表スペースまたは DMS 表スペースのどちらでもかまいませんが、 SYSTEM TEMPORARY 表スペースのページ・サイズは、元の索引表スペースのページ・サイズに一致しなければなりません。

ロード操作が **ALLOW READ ACCESS** モードでない場合、または索引付けモードに互換性がない場合には、 **USE** *tablespace-name* オプションは無視されます。 **USE** *tablespace-name* オプションは、**INDEXING MODE REBUILD** オプションまたは **INDEXING MODE AUTOSELECT** オプションでのみサポートされます。**INDEXING MODE AUTOSELECT** オプションが指定されており、ロード・ユーティリティーが索引の増分保守を選択する場合には、**USE** *tablespace-name* は無視されます。

ロード再開操作では、元のロード操作で代替表スペースを使用しなかった場合でも、代替表スペースを使用して索引を作成できます。元のロード操作が **ALLOW READ ACCESS** モードで発行されなかった場合には、 **ALLOW READ ACCESS** モードでロード再開操作を発行することはできません。ロード終了操作では索引を再作成しないため、 **USE** *tablespace-name* は無視されます。

ロード操作の構築フェーズでは、 SYSTEM TEMPORARY 表スペースに索引が作成されます。その後、索引コピー・フェーズで、 SYSTEM TEMPORARY 表スペースから元の索引表スペースに索引がコピーされます。元の索引表スペースに新規索引用の十分なスペースがあることを確認するには、構築フェーズで元の表スペースにスペースを割り振らなければなりません。したがって、ロード操作で索引スペースが不足する場合には、構築フェーズでこれを実行します。この場合、元の索引が消失することはありません。

索引コピー・フェーズは、構築および削除フェーズの後に実行されます。索引コピー・フェーズが始まる前に、表が排他的にロックされます。つまり、索引コピー・フェーズ全体に渡って、読み取りアクセスは使用不可になります。索引コピー・フェーズは物理コピーであるため、表はかなりの期間使用できなくなります。

注: SYSTEM TEMPORARY 表スペースまたは索引表スペースのどちらかが DMS 表スペースである場合、 SYSTEM TEMPORARY 表スペースの読み取りにより、SYSTEM TEMPORARY 表スペースでのランダム入出力が発生し、そのために遅延が生じる可能性があります。索引表スペースへの書き込みはこれまでどおり最適化され、**DISK_PARALLELISM** 値が使用されます。

大規模な索引に関する考慮事項

ロード時の大規模な索引作成のパフォーマンスの改善には、**sortheap** データベース構成パラメーターの調整が役に立つことがあります。 **sortheap** は、ロード操作中に索引キーのソート処理専用に割り振るメモリーの大きさを指定します。例えば、キーのソート処理で索引ごとに 4000 ページの主メモリーを使用するようロード・ユーティリティーに指示するには、**sortheap** を 4000 ページに設定し、データベースからすべてのアプリケーションを切り離した後、**LOAD** コマンドを発行します。

索引が大きすぎてメモリー内でソートできないと、ソート・スピルあるいはオーバーフローが起こります。つまり、データは、複数の「ソート実行」で分割され、後で組み合わせられる TEMPORARY 表スペースに保管されます。ソート・スピルが発生したかどうかを判別するには、**sort_overflows** モニター・エレメントを使用します。 **sortheap** パラメーターのサイズを大きくしてもソート・スピルを避けられない場合、 TEMPORARY 表スペースのバッファー・プールを十分大きくして、スピルが原因で生じるディスク入出力量を最小化することを確認してください。さらに、複数のソート実行の組み合わせにおいて入出力並列処理を実現するため、 TEMPORARY 表スペースの宣言時に、それぞれが異なるディスク装置に存在する複数のコンテナーを指定することをお勧めします。ロード操作はすべてのキーをメモリー内に維持するため、表に複数の索引が定義されている場合は、それに比例してメモリーの消費量が増加します。

索引作成の据え置き

一般的に言えば、索引作成を据え置くよりも、REBUILD または INCREMENTAL モードのいずれかを指定することによりロード操作中に索引を作成する方が効率的です。 図 11 で示すように、多くの場合、表はデータのロード、索引の構築、統計の収集という 3 つのステップによって構築されます。このため、ロード操作中、索引の作成中 (表ごとに複数の索引が可能)、および統計の収集中 (表データと索引すべてに対する入出力が発生) に、複数のデータ入出力が発生することになります。別の方法として、ロード・ユーティリティーがデータを一括してこれらの作業を実行するようにすれば、さらに時間が短くなります。ただし、ユニーク索引がある場合、重複データが検出されるとロードのパフォーマンスは低下します。

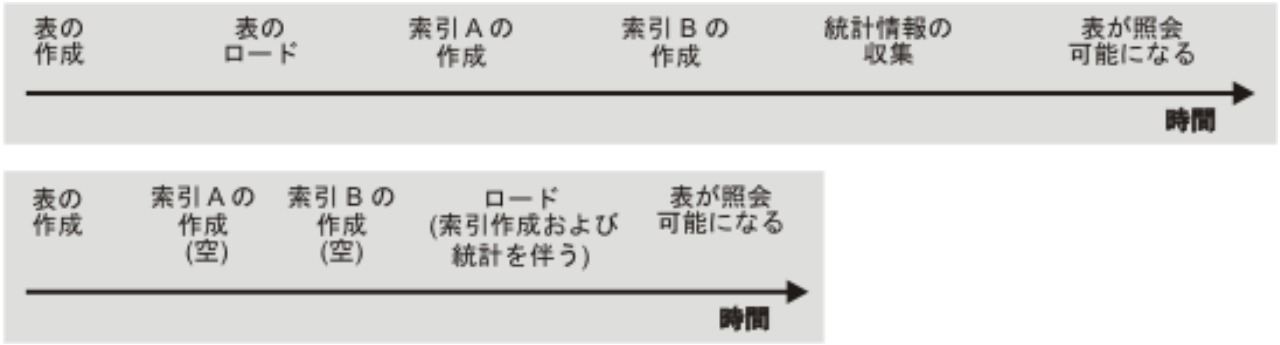


図 11. 索引作成と統計収集の並行処理によるロード・パフォーマンスの向上： 多くの場合、表はデータのロード、索引の構築、統計の収集という 3 つのステップによって構築されます。このため、ロード操作中、索引の作成中 (表ごとに複数の索引が可能)、および統計の収集中 (表データと索引すべてに対する入出力が発生) に、複数のデータ入出力が発生することになります。別の方法として、ロード・ユーティリティーがデータを一括してこれらの作業を実行するようにすれば、さらに時間が短くなります。

場合によっては、索引作成を据え置き、CREATE INDEX ステートメントを呼び出すことでパフォーマンスが改善されることもあります。索引再作成時のソートが使用するページの最大数は **sortheap** で指定されています。さらにスペースが必要な場合は、TEMP バッファー・プールが使用され、(最終的には) ディスクにスピルされます。ロードがスピルされ、結果としてパフォーマンスが低下する場合には、**INDEXING MODE DEFERRED** を使用して **LOAD** を実行し、索引を後で再作成したほうが賢明な場合があります。 CREATE

INDEX は、一度に 1 つの索引を作成するので、表を何度もスキャンしてキーを収集するのに比べてメモリーの使用量を削減できます。

ロード操作と同時に索引を作成する代わりに CREATE INDEX ステートメントを使用して索引を作成するもう 1 つの利点は、CREATE INDEX ステートメントが複数のプロセス (つまりスレッド) を使用してキーをソートできる点です。実際の索引の構築は、並列で実行されません。

ロード操作時のコンプレッション・ディクショナリーの作成

圧縮が有効になっている表で **LOAD INSERT** 操作および **LOAD REPLACE** 操作を行うと、コンプレッション・ディクショナリーの作成をトリガーすることができます。ディクショナリーの作成方法は、表が使用する行圧縮のタイプに応じて異なります。

クラシック行圧縮 では、単一の*表レベルのコンプレッション・ディクショナリー*を使用して、データを圧縮していました。*アダプティブ圧縮* では、クラシック行圧縮で使用されていた表レベルのコンプレッション・ディクショナリーとともに、複数の*ページ・レベルのコンプレッション・ディクショナリー* を使用してデータの個々のページを圧縮します。

ページ・レベルのコンプレッション・ディクショナリー

ページ・レベルのディクショナリーは、**LOAD INSERT** 操作または **LOAD REPLACE** 操作の際に、自動的に作成および更新されます。**LOAD** コマンドの **KEEPDICTIONARY** オプションと **RESETDICTIONARY** オプションは、ページ・レベルのディクショナリーに影響を与えません。

表レベルのコンプレッション・ディクショナリー

ディクショナリーが存在しない場合、表レベルのディクショナリーは LOAD INSERT 操作と LOAD REPLACE 操作のどちらの場合にも自動的に作成されます。ただし、表レベルのディクショナリーが存在する場合、デフォルトで、ディクショナリーは更新されません。より具体的に言うと、**LOAD REPLACE** 操作はデフォルトで **KEEPDICTIONARY** オプションを想定します。 **RESETDICTIONARY** オプションを指定することで、既存の表レベルのディクショナリーを削除し、新しいものを作成することができます。

LOAD INSERT は常に、**KEEPDICTIONARY** オプションによって暗黙指定される動作に従います。

非 XML データの表レベルのディクショナリーを作成する場合、ロード・ユーティリティーは、ターゲット表に存在するデータを使用してディクショナリーを作成します (この既存のデータはその表に保管されるデータの種類を代表するものであると想定)。ターゲット表の既存のデータが不足している場合、ロード・ユーティリティーは十分な入力データをサンプリングしてから、ディクショナリーを作成します。この場合、ロード・ユーティリティーは、入力データのみを使用してディクショナリーを作成します。

XML データの場合、ロード・ユーティリティーは入力データだけをサンプリングします。

範囲パーティション表でディクショナリーが作成されると、パーティションはそれぞれ個々の表のように扱われます。ディクショナリーがパーティション間で共有されることはなく、既にディクショナリーが存在するパーティションではディクショナリーの作成は行われません。表データの場合、各パーティションで生成されるディクショナリーはそのパーティションだけの既存の表データ (および、必要な場合はロードされるデータ) に基づきます。バージョン 9.7 フィックスパック 1 以降では、パーティションにある既存のデータが最小しきい値より少ない場合、ロードされるデータのみに基づいてディクショナリーを作成します。XML データの場合、各パーティションで生成されるディクショナリーは、そのパーティションにロードされるデータに基づきます。

KEEPDICTIONARY オプションを使用した LOAD REPLACE

KEEPDICTIONARY オプションを使用する **LOAD REPLACE** は既存のディクショナリーを維持し、ターゲット表の COMPRESS 属性が有効になっている限り、そのディクショナリーを使用して、ロードされたデータを圧縮します。ディクショナリーが存在しない場合、ロード・ユーティリティーは COMPRESS 属性が有効になっている表に対して新規のディクショナリーを生成します (ただし、表にロードされているデータが表の行またはデフォルトの XML ストレージ・オブジェクトに保管された XML 文書の、事前に決定されているしきい値を超える場合に限ります)。ターゲット表のデータは置き換えられるので、ロード・ユーティリティーは入力データのみを使ってディクショナリーを作成します。ディクショナリーの作成後、そのディクショナリーは表に挿入され、ロード操作が継続します。

RESETDICTIONARY オプションを使用した LOAD REPLACE

COMPRESS 属性がオンになっている表にロードする際に **RESETDICTIONARY** オプションを使用することには 2 つの重要な含意があります。まず、**LOAD REPLACE** が完了した後で、ターゲット表に少しでもデータが存在していれば、ディクショナリーの作成が発生します。つまり、新規のコンプレッション・ディクショナリーは、1 行のデータまたは 1 つの XML 文書に基づいて作成される場合もあるということです。また、次のいずれかの状況が存在する場合、既存のディクショナリーは置換ではなく削除される (ターゲット表はコンプレッション・ディクショナリーを失う) ということにもなります。

- COMPRESS 属性がオフになっている表に対して操作が実行される場合。
- 何もロードされない場合 (ゼロ行)。この場合、通知ログに ADM5591W が出力されます。

注: **RESETDICTIONARY** オプションを指定した **LOAD REPLACE** 後に **LOAD TERMINATE** 操作を実行する場合、既存のコンプレッション・ディクショナリーは置換ではなく削除されます。

パフォーマンスへの影響

ディクショナリーの作成はロード操作のパフォーマンスに次の 2 つの点で影響を与えます。

- **LOAD INSERT** 操作は、表レベルのコンプレッション・ディクショナリーを作成する前に、ディクショナリー作成の最小しきい値だけでなく既存の表データすべてをスキャンします。そのため、表のサイズが大きければ大きいほどこのスキャンに時間がかかります。
- コンプレッション・ディクショナリーを作成するための追加処理がありますが、ディクショナリーの作成にかかる時間は最小です。

ディクショナリーの作成に関連する操作が **LOAD** コマンドによる CPU 使用率に影響を与えることがありますが、ロード操作には通常、入出力制約があります。つまり、ロードの完了までの待機に費やされる時間の多くは、ディスクへのデータの書き込みを待機する時間です。一般的には、ディクショナリーの作成によって CPU のロードが増えても、ロードの実行に必要な経過時間が増えることはありません。実際には、データは圧縮形式で書き込まれるため、圧縮解除された表にデータをロードする場合と比べて入出力時間は減ることがあります。

ロード・パフォーマンスを改善するためのオプション

ロード・パフォーマンスを最適化するために使用できる各種コマンド・パラメーターがあります。さらに、ロード固有のファイル・タイプ修飾子もいくつかあり、それらは時としてユーティリティーのパフォーマンスを著しく向上させる場合があります。

コマンド・パラメーター

DISK_PARALLELISM、CPU_PARALLELISM、および DATA BUFFER の各パラメーターがユーザーによって指定されていない場合、ロード・ユーティリティーはこれらのパラメーターの最適な値を決定することにより、パフォーマンスを最大にするよう試みます。最適化は、ユーティリティー・ヒープにおける使用可能なサイズとフリー・スペースに基づいてなされます。これらのパラメーターを特定の必要に合わせて調整しようとする前に、オートノミック DISK_PARALLELISM および CPU_PARALLELISM 設定の使用を考慮してください。

以下に、ロード・ユーティリティーで利用可能な各種オプションとパフォーマンスとの関係を説明します。

ALLOW READ ACCESS

このオプションを使用すると、ロード操作の進行中に表を照会することができます。ロード操作前に表に存在していたデータのみ表示できます。INDEXING MODE INCREMENTAL オプションも指定されているときに、ロード操作が失敗した場合には、後続のロード終了操作で、索引での矛盾を訂正する必要があります。これを行うには、相当量の入出力を伴う索引スキャンが必要になります。ロード終了操作に ALLOW READ ACCESS オプションも指定されている場合には、入出力にバッファー・プールが使用されます。

重要: バージョン 10.1 フィックスパック 1 以降、ALLOW READ ACCESS パラメーターは非推奨となっており、将来のリリースで除去される可能性があります。詳しくは、「*DB2 バージョン 10.1 の新機能*」の『LOAD コマンドの ALLOW READ ACCESS パラメーターが非推奨になった』を参照してください。

COPY YES または NO

このパラメーターは、ロード操作中に入力データのコピーを作成するかどう

かを指定するのに使用します。 COPY YES (順方向リカバリーが有効になっている場合のみ適用可能) を指定すると、ロードするデータはすべてロード操作中にコピーされるため、ロードのパフォーマンスは低下します。入出力活動が増加すると、入出力制約のシステムにおけるロード時間が大きくなることがあります。複数の装置やディレクトリー (異なるディスク上にある) を指定することにより、この操作によって生じるパフォーマンス上の不利をいくらか相殺できます。 COPY NO (順方向リカバリーが有効になっている場合のみ適用可能) を指定すると、ロード・パフォーマンスに影響はありません。ただし、ロードされる表に関連した表スペースはすべて「バックアップ・ペンディング」状態になり、それらの表スペースをバックアップしてからでないと表にアクセスできなくなります。

CPU_PARALLELISM

このパラメーターは、データベース・パーティション当たりの実行プロセス数 (マシンが対応している場合) を利用してロードのパフォーマンスを大幅に向上させたい場合に使用します。このパラメーターには、ロード・ユーティリティーがデータ・レコードの構文解析、変換、およびフォーマット設定に使用するプロセス数またはスレッド数を指定します。指定可能な最大数は 30 です。指定された値をサポートするためのメモリーが十分でない場合、ユーティリティーはこの値を調整します。このパラメーターを指定しない場合、ロード・ユーティリティーはシステムの CPU 数に基づくデフォルト値を選択します。

パラメーターの値に関係なく、ソース・データ内のレコードの順序は保持されます (図 12を参照)。ただし、次の条件を満たしている必要があります。

- anyorder ファイル・タイプ修飾子が指定されていない
- PARTITIONING_DBPARTNUMS オプションで複数のパーティションが指定されていない

表に LOB または LONG VARCHAR のいずれかのデータが格納されていると、CPU_PARALLELISM は 1 に設定されます。この場合、並列処理はサポートされません。

このパラメーターの使用は対称マルチプロセッサー (SMP) ハードウェアに限定されていませんが、非 SMP 環境でこのパラメーターを使っても、明確なパフォーマンスの向上は期待できません。

図 12. ロード操作中に、データベース・パーティション当たりの実行プロセス数を利用した場合、ソース・データのレコード順序は保持される

DATA BUFFER

DATA BUFFER パラメーターは、ロード・ユーティリティーにバッファーとして割り当てるメモリーの合計を指定します (4 KB 単位)。このバッファーは、サイズに応じていくつかのエクステント にしておくことをお勧めします。データ・バッファーはユーティリティー・ヒープから割り当てられます

が、システムに使用可能メモリーがある場合は、 *util_heap_sz* データベース構成パラメーターの設定値を超えることができます。

DISK_PARALLELISM

DISK_PARALLELISM パラメーターは、ロード・ユーティリティーがデータ・レコードをディスクに書き込むのに使用するプロセス数またはスレッド数を指定します。このパラメーターは、データのロード時に使用可能なコンテナーを利用してロードのパフォーマンスを大幅に向上させたい場合に使用します。指定可能な最大数は、CPU_PARALLELISM 値 (ロード・ユーティリティーが実際に使用している値) の 4 倍か 50 のいずれか大きい方です。デフォルトでは、 DISK_PARALLELISM はロードする表のオブジェクトを備えたすべての表スペース上にある表スペース・コンテナーの合計数と等しい値です (この値が指定可能な最大数を超えていない場合)。

NONRECOVERABLE

順方向リカバリーが有効になっており、ロールフォワード時に表に対するロード・トランザクションをリカバリーする必要がない場合にこのパラメーターを使用します。 NONRECOVERABLE ロードのパフォーマンスと COPY NO ロードのパフォーマンスは同じです。しかし、データ損失の可能性に関しては大きな違いがあります。 NONRECOVERABLE ロードは、表に完全にアクセスできるようにする一方で、その表をロールフォワード・リカバリー不可能としてマークします。これによって問題が生じる場合があります。ロード操作によってロールフォワードする必要がある場合、ロードされたデータおよび表に対するそれ以降の更新がすべて失われる可能性があります。 COPY NO ロードは、従属表スペースをすべて「バックアップ・ペンディング」状態にして、バックアップが実行されるまで表にアクセスできないようにします。その種のロードの後にはバックアップが強制されるため、ロードされたデータまたは表に対してそれ以降行われる更新が失われる危険はありません。つまり、COPY NO ロードを使用すると完全なリカバリーが行えます。

注: これ以後のリストアやロールフォワード・リカバリー操作の間にそれらのロード・トランザクションが検出された場合、表は更新されずに invalid としてマークされます。それ以降、この表に対する処理は無視されます。ロールフォワード操作の完了後は表のドロップのみが可能です。

SAVECOUNT

このパラメーターは、ロード操作の**ロード・フェーズ中** に整合点を確立するインターバルを設定するのに使用します。整合点を確立するために実行される活動を同期化するには時間がかかります。これをあまりに頻繁に実行すると、ロードのパフォーマンスがかなり低下します。大量の行をロードすることになっている場合は、SAVECOUNT 値を大きく指定することをお勧めします (例えば 1 億個のレコードが関係するロード操作の場合は値 10,000,000 など)。

ロード再開操作は、最後の整合点から自動的に続行します。ただし、ロード・フェーズから再開する場合に限ります。

STATISTICS USE PROFILE

表統計プロファイルで指定した統計を収集します。このパラメーターを使用すると、ロード操作の完了後に RUNSTATS ユーティリティーを呼び出す場

合よりも効率よくデータ分散と索引統計情報を収集することができます。ただし、ロード操作そのもののパフォーマンスは低下します (特に DETAILED INDEXES ALL を指定した場合)。

最適なパフォーマンスを得るために、アプリケーションには入手可能な最大限のデータ分散と索引統計情報が必要です。統計情報が更新されると、アプリケーションではその最新の統計情報に基づいて表データへの新しいアクセス・パスを使用できます。表への新しいアクセス・パスは、**BIND** コマンドを使ってアプリケーション・パッケージを再バインドすることにより作成できます。表統計プロファイルは、SET PROFILE オプションを指定して **RUNSTATS** コマンドを実行することによって作成されます。

データを大規模な表にロードする場合は、*stat_heap_sz* (統計ヒープのサイズ) データベース構成パラメーターに指定する値をさらに大きくすることをお勧めします。

USE <tablespace-name>

ALLOW READ ACCESS ロードが行われており、索引付けモードが REBUILD である場合にこのパラメーターを使用すると、 SYSTEM TEMPORARY 表スペースで索引を再構築し、ロード操作の索引コピー・フェーズで索引表スペースにコピーし直すことができます。

デフォルトでは、完全に再構築された索引 (*シャドー索引とも呼ばれる*) は、元の索引と同じ表スペースに作成されます。この場合、元の索引とシャドー索引の両方が同じ表スペースに同時に置かれているため、リソース問題の原因となる可能性があります。シャドー索引が元の索引と同じ表スペースに作成される場合には、元の索引は即座にシャドーに置き換えられます。ただし、SYSTEM TEMPORARY 表スペースにシャドー索引が作成される場合には、ロード操作で索引コピー・フェーズが必要になります。このフェーズでは、SYSTEM TEMPORARY 表スペースから索引表スペースに索引をコピーします。このコピーに関連した入出力はかなり大きくなります。表スペースのいずれかが DMS 表スペースである場合、 SYSTEM TEMPORARY 表スペースの入出力の順序が変わる可能性があります。 DISK_PARALLELISM オプションにより指定される値は、索引コピー・フェーズで優先されます。

WARNINGCOUNT

このパラメーターには、ロード操作を強制終了するまでにユーティリティーが戻すことのできる警告の数の限界値を指定します。警告が少ししかない、あるいはまったくないことが予想される場合、 WARNINGCOUNT パラメーターを比較的低い数値に設定してください。 WARNINGCOUNT の数に達すると、ロード操作は停止します。これを指定することにより、ロード操作を完了する前に問題を訂正することが可能になります。

ファイル・タイプ修飾子

ANYORDER

デフォルトでは、ロード・ユーティリティーはソース・データのレコード順序を保持します。 SMP 環境でロードが行われるときにその順序を保持するには、並列処理間で同期する必要があります。

SMP 環境で anyorder ファイル・タイプ修飾子を指定すると、順序を保持しないという指示がロード・ユーティリティーに与えられます。これによ

り、その順序を保持するために必要な同期をしないですむため、効率が上がります。しかし、ロードするデータがあらかじめソートされている場合、anyorder を指定するとその順序が崩れてしまい、あらかじめソートしておくことによるメリットがそれ以降の照会で失われてしまいます。

注: CPU_PARALLELISM が 1 の場合、anyorder ファイル・タイプ修飾子は何の影響も及ぼしません。また、この修飾子には SAVECOUNT オプションとの互換性がありません。

BINARYNUMERICS、ZONEDDECIMAL、および PACKEDDECIMAL

固定長の区切りなし ASCII (ASC) ソース・データの場合、数値データをバイナリーで表現するとロード時のパフォーマンスが向上します。packeddecimal ファイル・タイプ修飾子が指定される場合、ロード・ユーティリティーは 10 進データをパック 10 進数フォーマット (バイトにつき 2 桁) として解釈します。 zoneddecimal ファイル・タイプ修飾子が指定される場合、ロード・ユーティリティーは 10 進データをゾーン 10 進フォーマット (バイトにつき 1 桁) として解釈します。それ以外の数値タイプの場合はすべて、binarynumerics ファイル・タイプ修飾子が指定されると、ロード・ユーティリティーはデータをバイナリー・フォーマットとして解釈します。

注:

- binarynumerics、packeddecimal、または zoneddecimal ファイル・タイプ修飾子が指定されると、プラットフォームに関係なく、数値データはビッグ・エンディアン (最上位のバイトから記録/送信する) フォーマットとして解釈されます。
- packeddecimal ファイル・タイプ修飾子と zoneddecimal ファイル・タイプ修飾子を同時に指定することはできません。
- packeddecimal および zoneddecimal ファイル・タイプ修飾子は 10 進ターゲット列に対してのみ適用され、バイナリー・データはターゲット列の定義と一致しなければなりません。
- binarynumerics、packeddecimal、または zoneddecimal ファイル・タイプ修飾子を指定するときは、reclen ファイル・タイプ修飾子も指定する必要があります。

FASTPARSE

使用に際しては、注意が必要です。ロードされているデータが有効であることが明確な場合は、注意が必要なデータのロードを行う場合に比べ、徹底した構文検査をロードの際に実行する必要はない可能性があります。実際、このステップの有効範囲を狭めることによって、約 10% から 20% ほどロードのパフォーマンスを向上させることができます。これは fastparse ファイル・タイプ修飾子を使用することによって行えます。この修飾子は ASC および DEL ファイルのユーザー指定の列値に対して実行されるデータ・チェックを簡略化します。

NOROWWARNINGS

ロード操作中にリジェクトされた行に関する警告メッセージは、指定されたファイルに書き込まれます。しかし、リジェクトされたレコード、無効なレコード、または切り捨てられたレコードをロード・ユーティリティーが大量

に処理する必要がある場合は、ロードのパフォーマンスに対してマイナスの影響を与える可能性があります。多数の警告が予期されるような場合は、norowwarnings ファイル・タイプ修飾子を使用してこれらの警告の記録を抑制すると良いでしょう。

PAGEFREESPACE、INDEXFREESPACE、および TOTALFREESPACE

データを表に挿入し、更新していくうちに、表や索引の再編成が必要になります。 1 つの解決策は、pagefreespace、indexfreespace、および totalfreespace を使って表および索引のフリー・スペースの量を増やすことです。最初の 2 つの修飾子は PCTFREE の値に優先し、フリー・スペースとして残されるデータおよび索引ページのパーセンテージを指定します。一方、totalfreespace はフリー・スペースとして表に追加される総ページ数のパーセンテージを指定します。

参照整合性を維持するためのロードのフィーチャー

一般的に、ロード・ユーティリティーはインポート・ユーティリティーより効率的なユーティリティーですが、ロードされる情報の参照整合性を保つためにさまざまなフィーチャーを必要とします。

- **表ロック**。これは並行性制御を提供し、ロード操作中、データ・アクセスが無制限に行われないようにします。
- **表の状態** および **表スペースの状態**。これを使用すると、データへのアクセスを制御するか、または特定のユーザー処置を引き出すことができます。
- **ロード例外表**。知らずに無効データの行が削除されることがないようにします。

ロード操作に続く整合性違反のチェック

以下のいずれかの状態が存在する場合は、ロード操作の後に、その表が READ または NO ACCESS モードで SET INTEGRITY ペンディング状態になっていることがあります。

- 表に表チェック制約または参照整合性制約が定義されている場合。
- この表に生成列があり、 V7 以前のクライアントを使用してロード操作が開始された場合。
- この表に従属する IMMEDIATE 指定のマテリアライズ照会表またはこの表を参照する IMMEDIATE 指定のステージング表がある場合。
- 表がステージング表またはマテリアライズ照会表の場合。

ロードした表の SET INTEGRITY ペンディング状態は、その表に対応する SYSCAT.TABLES 項目の STATUS フラグに示されます。 STATUS の値が N で、また ACCESS MODE の値が F である場合に、ロードした表が完全に使用可能となります。これは表が完全にアクセス可能であり、通常状態であることを示します。

ロードされる表に従属表がある場合には、SET INTEGRITY PENDING CASCADE パラメーターを指定して、ロードされる表の SET INTEGRITY ペンディング状態が即時に従属表にカスケードされるようにするかどうかを指示できます。

ロードされる表に、従属外部キー表、従属マテリアライズ照会表、および従属ステージング表と共に制約があるときに、すべての表がロード操作前に通常状態である場合には、指定されるロード・パラメーターに基づいて、以下のような結果になります。

INSERT、ALLOW READ ACCESS、および SET INTEGRITY PENDING CASCADE IMMEDIATE

ロードされる表、その従属マテリアライズ照会表、および従属ステージング表は、読み取りアクセスを持つ SET INTEGRITY ペンディング状態になります。

INSERT、ALLOW READ ACCESS、および SET INTEGRITY PENDING CASCADE DEFERRED

ロードされる表だけが読み取りアクセスを持つ SET INTEGRITY ペンディング状態に置かれます。従属外部キー表、従属マテリアライズ照会表、および従属ステージング表は元の状態のままです。

INSERT、ALLOW NO ACCESS、および SET INTEGRITY PENDING CASCADE IMMEDIATE

ロードされる表、その従属マテリアライズ照会表、および従属ステージング表は、アクセスを持たない SET INTEGRITY ペンディング状態になります。

INSERT または REPLACE、ALLOW NO ACCESS、および SET INTEGRITY PENDING CASCADE DEFERRED

ロードされる表だけが、アクセスを持たない SET INTEGRITY ペンディング状態になります。従属外部キー表、IMMEDIATE 指定の従属マテリアライズ照会表、および IMMEDIATE 指定の従属ステージング表は元の状態のままです。

REPLACE、ALLOW NO ACCESS、および SET INTEGRITY PENDING CASCADE IMMEDIATE

表およびそのすべての従属外部キー表、IMMEDIATE 指定の従属マテリアライズ照会表、および IMMEDIATE 指定の従属ステージング表は、アクセスを持たない SET INTEGRITY ペンディング状態になります。

注: ロード置換操作で ALLOW READ ACCESS オプションを指定すると、エラーが発生します。

SET INTEGRITY ペンディング状態を除去するには、SET INTEGRITY ステートメントを使用します。 SET INTEGRITY ステートメントは表をチェックして制約違反がないかどうかを調べ、その表の SET INTEGRITY ペンディング状態を終了します。すべてのロード操作が INSERT モードで実行される場合、SET INTEGRITY ステートメントを使用して制約を増分的に処理します (つまり表のうち追加された部分だけをチェックして、制約違反がないかどうかを調べます)。例えば、以下のようにします。

```
db2 load from infile1.ixf of ixf insert into table1
db2 set integrity for table1 immediate checked
```

制約違反がないかどうかをチェックするのは TABLE1 のうち追加部分だけです。追加部分だけをチェックして制約違反がないかどうかを調べることにより、表全体をチェックするよりも時間が短くてすみます。これは、大きな表にデータを少しだけ追加した場合に特に有効です。

IBM Data Studio バージョン 3.1 以降では、次のタスクのためにタスク・アシスタントを使用できます: 整合性の設定. タスク・アシスタントは、オプションの設定、

タスク実行のために自動生成されたコマンドの確認、およびそれらのコマンドの実行のプロセスをガイドします。詳しくは、タスク・アシストを使用したデータベースの管理を参照してください。

SET INTEGRITY PENDING CASCADE DEFERRED オプションを指定して表がロードされるときに、 SET INTEGRITY ステートメントを使用して整合性違反をチェックする場合には、従属表はアクセスを持たない SET INTEGRITY ペンディング状態になります。表をこの状態から解除する場合には、明示的要求を発行する必要があります。

従属マテリアライズ照会表または従属ステージング表を持つ表が INSERT オプションを使用してロードされるときに、SET INTEGRITY ステートメントを使用して整合性違反をチェックする場合には、表は SET INTEGRITY ペンディング状態ではなくなり、No Data Movement モードに入れられます。これは、従属マテリアライズ照会表の後続の増分リフレッシュ、および従属ステージング表の増分伝搬を容易にするために実行されます。 No Data Movement 状態では、表内の行の移動の原因となるような操作は許可されません。

No Data Movement 状態は、SET INTEGRITY ステートメントを発行する際に FULL ACCESS オプションを指定することによりオーバーライドできます。表は完全にアクセス可能になりますが、従属マテリアライズ照会表の完全再計算が後続の REFRESH TABLE ステートメントで実行され、従属ステージング表は不完全な状態になります。

ロード操作で ALLOW READ ACCESS オプションが指定されている場合には、SET INTEGRITY ステートメントを使用して制約違反がチェックされるまで、表は読み取りアクセス状態のままです。ロード操作がコミットされたら、アプリケーションは表で、ロード操作前に存在したデータを照会できますが、SET INTEGRITY ステートメントが発行されるまでは、新しくロードされたデータを表示することはできません。

制約違反をチェックする前に、いくつかのロード操作を表で実行できます。ALLOW READ ACCESS モードですべてのロード操作が完了した場合には、最初のロード操作の前に表に存在していたデータだけが照会に使用できます。

表が 1 つでも複数でも、このステートメントを 1 回呼び出すだけでチェックできます。従属表を独自にチェックする場合、その親表が SET INTEGRITY ペンディング状態になっていてはなりません。そうしないと、親表と従属表の両方を同時にチェックしなければならなくなります。参照整合性が循環している場合、その循環に関係しているすべての表を 1 回の SET INTEGRITY ステートメント呼び出しに組み込む必要があります。従属表をロードしている間に、親表に制約違反がないかどうかをチェックするのがよいかもしれません。ただしこれが可能なのは、2 つの表が同じ表スペースにない場合だけです。

SET INTEGRITY ステートメントの発行時に INCREMENTAL オプションを指定すると、増分処理を明示的に要求することができます。しかし、ほとんどの場合 DB2 データベースは増分処理を選択するため、このオプションは不要です。増分処理を実行できない場合には、自動的に全処理が実行されます。 INCREMENTAL オプションを指定したにもかかわらず増分処理を実行できない場合、以下に示す条件が成立するならエラーが戻されます。

- 表が SET INTEGRITY ペンディング状態の間に、この表に新しく制約が追加される場合。
- 表に対する最後の整合性チェックの後で、ロード置換操作が行われるか、または NOT LOGGED INITIALLY WITH EMPTY TABLE オプションが活動化される場合。
- 親表が、ロード置換されるか、または増分ではない方法で整合性チェックされる場合。
- 表が、アップグレード前の SET INTEGRITY ペンディング状態にある場合。アップグレード後に最初に表が整合性チェックされるときは、完全処理が必要。
- 表またはその親表の入った表スペースがある時点でロールフォワードされ、表とその親が異なる表スペースに配置される場合。

表で SYSCAT.TABLES カタログの CONST_CHECKED 列に 1 つまたは複数の W 値があるときに、SET INTEGRITY ステートメントに NOT INCREMENTAL オプションが指定されていない場合、表は増分処理され、SYSCAT.TABLES の CONST_CHECKED 列には、すべてのデータをシステムがチェックしたわけではないことを示す U というマークが付けられます。

SET INTEGRITY ステートメントは、制約違反のある行を削除した結果として DELETE トリガーを起動することはありませんが、表が SET INTEGRITY ペンディング状態ではなくなるとトリガーが起動されます。そのため、例外表のデータを収集して、ロードした表に例外表の行を挿入すると、その表に定義されている INSERT トリガーが起動されます。これについては考慮が必要です。 1 つの選択肢は、INSERT トリガーをいったんドロップしてから例外表から行を挿入し、その後で INSERT トリガーを再作成することです。

ロード操作時の表のロッキング

大抵の場合、ロード・ユーティリティーは、表レベル・ロックを使用して、表へのアクセスを制限します。ロックのレベルは、ロード操作の段階、およびロード操作が読み取りアクセスを許可するように指定されているかどうかによって異なります。

ALLOW NO ACCESS モードのロード操作は、ロード中に表に対して超排他ロック (Z-lock) を使用します。

ALLOW READ ACCESS モードのロード操作を開始する前に、ロード・ユーティリティーは、ロード操作前に開始したすべてのアプリケーションが、ターゲット表に対するロックを解放するのを待機します。ロード操作の始めに、ロード・ユーティリティーは表に対する更新ロック (U ロック) を獲得します。これは、データがコミットされるまで、このロックを保留します。ロード・ユーティリティーが表に対する U ロックを獲得するとき、ロード操作の開始前に表に対するロックを保持するすべてのアプリケーションがそれらのロック (互換性のあるロックでも) を解放するのを待機します。これは U ロックを Z ロックに一時的にアップグレードすることによって達成されます。ターゲット表に新しく表ロック要求が出されても、要求されるロックがロード操作の U ロックと互換性のあるものである限り、Z ロックがこれと競合することはありません。データがコミットされるときに、ロード・ユーティリティーはロックを Z ロックにアップグレードするため、コミット時には、競合す

るロックを持つアプリケーションが終了するまでロード・ユーティリティーが待機することで、いくらかの遅延が発生する場合があります。

注: アプリケーションが表に対するロックを解放するのを待機する間に、ロード操作がロードを開始しないうちにタイムアウトになる可能性があります。ただし、データをコミットするために必要な Z ロックを待機している間に、ロード操作がタイムアウトになることはありません。

競合するロックを持つアプリケーション

LOAD コマンドの LOCK WITH FORCE オプションを使用すると、ターゲット表に対する競合するロックを保留するアプリケーションを強制的にオフにし、ロード操作を継続できるようにすることができます。 ALLOW READ ACCESS モードのロード操作を継続するためにまず、以下のロックを保留するアプリケーションが強制的にオフにされます。

- 表更新ロックと競合する表ロック (例: インポートまたは挿入)。
- ロード操作のコミット・フェーズで存在するすべての表ロック。

システム・カタログ表に対する競合するロックを保留するアプリケーションは、ロード・ユーティリティーによって強制的にオフにされることはありません。ロード・ユーティリティーによってアプリケーションが強制的にシステムからオフにされると、アプリケーションはそのデータベース接続を失い、エラーが戻されます (SQL1224N)。

リカバリー可能データベースのロード操作に COPY NO オプションが指定される場合、表スペースがバックアップ・ペンディング状態になるまで、ターゲット表スペースのすべてのオブジェクトが共有モードでロックされます。これは、アクセス・モードにかかわりなく発生します。 LOCK WITH FORCE オプションが指定されている場合には、共有ロックと競合する表スペースにあるオブジェクトに対してロックを保留するすべてのアプリケーションが強制的にオフになります。

読み取りアクセス・ロード操作

ロード・ユーティリティーは、他のアプリケーションがロード中の表に対して行うアクセスの量を制御する 2 つのオプションを提供します。 ALLOW NO ACCESS オプションは表を排他的にロックし、表のロード時には、表データへのアクセスを許可しません。

ALLOW NO ACCESS オプションはデフォルトの動作です。 ALLOW READ ACCESS オプションは、他のアプリケーションによる表へのすべての書き込みアクセスを行えないようにしますが、既存のデータへの読み取りアクセスは許可します。この項では、ALLOW READ ACCESS オプションを扱います。

重要: バージョン 10.1 フィックスパック 1 以降、ALLOW READ ACCESS パラメーターは非推奨となっており、将来のリリースで除去される可能性があります。詳しくは、「*DB2 バージョン 10.1 の新機能*」の『LOAD コマンドの ALLOW READ ACCESS パラメーターが非推奨になった』を参照してください。

ロード操作を開始する前に存在する表データおよび索引データは、ロード操作の進行中に照会で表示できます。次の例を考慮してください。

1. 1 つの整数列のある表を作成します。

```
create table ED (ed int)
```

2. 3 つの行をロードします。

```
load from File1 of del insert into ED
...
Number of rows read         = 3
Number of rows skipped      = 0
Number of rows loaded       = 3
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 3
```

3. 表を照会します。

```
select * from ED

ED
-----------
          1
          2
          3

  3 record(s) selected.
```

4. ALLOW READ ACCESS オプションを指定してロード操作を実行し、さらに 2 つの行のデータをロードします。

```
load from File2 of del insert into ED allow read access
```

5. 同時に、他の接続ではロード操作の進行中に表を照会します。

```
select * from ED

ED
-----------
          1
          2
          3

  3 record(s) selected.
```

6. ロード操作が終了するのを待ってから、表を照会します。

```
select * from ED

ED
-----------
          1
          2
          3
          4
          5

  5 record(s) selected.
```

ALLOW READ ACCESS オプションは、ロード操作が進行中の場合やロード操作が失敗した後でも、ユーザーがいつでも表データにアクセスできるようにするため、大量のデータをロードする際に大変便利です。 ALLOW READ ACCESS モードのロード操作の動作は、アプリケーションの分離レベルに依存しません。つまり、どの分離レベルであっても、事前に存在するデータを常に読み取ることができますが、ロード操作が終了するまでは新しくロードされたデータを読み取ることはできません。

ロード操作の最初と最後という 2 つの場合を除いては、その操作中ずっと読み取りアクセスが提供されます。

最初に、ロード操作はそのセットアップ・フェーズの終わり近くの少しの間、特殊 Z ロックを獲得します。この特殊 Z ロックを要求するロード操作より先に、表に対する非互換のロックを保持するアプリケーションがあった場合、ロード操作はこの非互換のロックが解放されるのを一定期間だけ待ち、その後タイムアウトになって失敗します。時間は *locktimeout* データベース構成パラメーターによって決定されます。 LOCK WITH FORCE オプションを指定すると、ロード操作は他のアプリケーションを強制的にオフにし、タイムアウトになるのを回避します。ロード操作は特殊 Z ロックを獲得し、このフェーズをコミットし、このロックを解放した後、ロード・フェーズに移ります。 ALLOW READ ACCESS モードでロード操作を開始した後、何らかのアプリケーションが表を読み取ろうとしてロックを要求した場合、ロックは付与されます。これが、この特殊 Z ロックと競合することはありません。新しいアプリケーションがターゲット表から既存のデータを読み取ろうとする場合に、これが可能です。

次に、ロード操作の最後でデータをコミットする前に、ロード・ユーティリティーは表に対する超排他ロック (Z ロック) を獲得します。ロード・ユーティリティーは、表に対してロックを保持しているすべてのアプリケーションがこれらを解放するまで待機します。これは、データがコミットされる前の遅延の原因となることがあります。 LOCK WITH FORCE オプションを使用すると、競合するアプリケーションを強制的にオフにし、待機せずにロード操作が継続されます。通常、ALLOW READ ACCESS モードのロード操作は、短期間の排他ロックを獲得します。しかし、USE <tablespace-name> オプションが指定されている場合には、索引コピー・フェーズの期間ずっと排他ロックが続きます。

複数のデータベース・パーティションで定義されている表に対してロード・ユーティリティーが実行している間、ロード・プロセス・モデルはそれぞれのデータベース・パーティションで個別に実行します。つまり、他のデータベース・パーティションからは独立してロックが獲得および解放されます。したがって、照会その他の操作が並行して実行された場合、同じロックに対して競合するため、デッドロックの可能性があります。例えば、データベース・パーティション 0 の表ロックが操作 A に認可され、データベース・パーティション 1 の表ロックがロード操作に認可されたとします。ロード操作がデータベース・パーティション 0 の表ロックを待って待機している間、操作 A はデータベース・パーティション 1 の表ロックが認可されるまで待機するため、デッドロックが発生する可能性があります。この場合、デッドロック検出機能は一方の操作を任意にロールバックします。

注:

1. ロード操作が中断または失敗すると、このロード操作の発行時に指定されたものと同じアクセス・レベルで継続します。したがって、ALLOW NO ACCESS モードのロード操作が失敗すると、ロードが終了するか、またはロード再開が発行されるまで、表データはアクセス不可になります。 ALLOW READ ACCESS モードのロード操作が打ち切られても、既存の表データはこれまでどおり読み取りアクセスが可能です。
2. 中断または失敗したロード操作に ALLOW READ ACCESS オプションが指定されていた場合には、ロード再開操作またはロード終了操作にもこれを指定することができます。しかし、中断または失敗したロード操作に ALLOW NO ACCESS オプションが指定されていた場合には、ロード再開操作またはロード終了操作に ALLOW READ ACCESS オプションを指定することはできません。

以下の場合には、ALLOW READ ACCESS オプションはサポートされません。

- REPLACE オプションが指定されている場合。ロード置換操作では、新規データをロードする前に既存の表データを切り捨てるため、ロード操作の完了後まで照会できる既存のデータはありません。
- 索引に無効のマークが付き、再作成されるのを待機している場合。一部のロールフォワード・シナリオで、または **db2dart** コマンドの使用時に、索引に無効のマークが付く可能性があります。
- INDEXING MODE DEFERRED オプションが指定されている場合。このモードでは、索引に再作成が必要というマークが付けられます。
- ALLOW NO ACCESS ロード操作が再開処理中または終了処理中の場合。これが完全にオンラインになるまで、表に対して ALLOW READ ACCESS モードのロード操作を実行することはできません。
- ロード操作が、SET INTEGRITY PENDING NO ACCESS 状態の表で実行されている場合。これは、制約付きの表に対する複数のロード操作でも同じです。 SET INTEGRITY ステートメントが発行されるまで、表がオンラインになることはありません。

一般に、表データがオフラインになっている場合には、この表がオンラインになるまでロード操作時に読み取りアクセスを使用することはできません。

ロード操作時およびロード操作後の表スペースの状態

ロード・ユーティリティーは、表スペースの状態を使用して、ロード操作時のデータベースの整合性を保持します。これらの状態は、データへのアクセスを制御するかユーザー処置を引き出すことによって機能します。

ロード・ユーティリティーは、ロード操作に関係する表スペースを静止させる (表スペースに対して永続ロックを置く) ことはなく、**COPY NO** パラメーターが指定されているロード操作の場合にのみ、表スペース状態を使用します。

表スペースの状態は、**LIST TABLESPACES** コマンドを使用することによって検査できます。表スペースは同時に複数の状態になる場合もあります。 **LIST TABLESPACES** によって戻される状態は以下のとおりです。

正常

「正常」という状態は、表スペースが作成された後の最初の状態であり、現在、表スペースに表れるような (異常な) 状態がないことを示しています。

ロード進行中

「ロード進行中」という状態は、表スペースに進行中のロードがあることを示しています。この状態にあると、ロード操作時に従属表のバックアップを行えません。ロード・ユーティリティーは、リカバリー可能データベースで **COPY NO** パラメーターを指定した場合のみ表スペースを「ロード進行中」の状態にするため、表スペースの状態は (すべてのロード操作で使用される) 「ロード進行中」の表の状態とは異なります。表スペースはロード操作の間はこの状態のままです。

バックアップ・ペンディング

リカバリー可能データベースに対してロード操作を実行し、**COPY NO** パラメーターを指定すると、表スペースは最初のコミット後、「バックアップ・ペ

ンディング」という表スペースの状態になります。「バックアップ・ペンディング」の状態にある表スペースは更新できません。表スペースの「バックアップ・ペンディング」状態を解除する唯一の方法は、表スペースをバックアップすることです。ロード操作をキャンセルしても、表スペースの状態はロード操作の開始時に変更されていてロールバックできないため、表スペースは引き続き「バックアップ・ペンディング」状態のままです。

リストア・ペンディング

COPY NO オプションを指定して正常なロード操作を実行し、データベースをリストアし、その操作によってロールフォワードを行った場合、関連する表スペースは「リストア・ペンディング」という状態になります。表スペースを「リストア・ペンディング」状態から解除するには、リストア操作を実行しなければなりません。

注: DB2 LOAD によって表スペース状態が**ロード・ペンディング**または**削除ペンディング**に設定されるわけではありません。

表スペースの状態の例

以下のようにして、入力ファイル (staffdata.del) を表 NEWSTAFF にロードするとします。

```
update db cfg for sample using logarchmeth1 logretain;
backup db sample;
connect to sample;
create table newstaff like staff;
load from staffdata.del of del insert into newstaff copy no;
connect reset;
```

さらに、別のセッションを開き、次のコマンドを発行します。

```
connect to sample;
list tablespaces;
connect reset;
```

USERSPACE1 (サンプル・データベースのデフォルトの表スペース) は「ロード進行中」の状態になり、最初のコミット後は「バックアップ・ペンディング」の状態にもなります。ロード操作の完了後、**LIST TABLESPACES** コマンドを実行することにより、USERSPACE1 が現在「バックアップ・ペンディング」の状態にあることが分かります。

```
Tablespace ID                        = 2
Name                                 = USERSPACE1
Type                                 = Database managed space
Contents                             = All permanent data. Large table space.
State                                = 0x0020
  Detailed explanation:
  Backup pending
```

ロード操作時およびロード操作後の表の状態

ロード・ユーティリティーは、表の状態を使用して、ロード操作時のデータベースの整合性を保持します。これらの状態は、データへのアクセスを制御するかユーザー処置を引き出すことによって機能します。

表の状態を判別するには、**LOAD QUERY** コマンドを発行します。このコマンドはロード操作の状態も検査します。表は同時に複数の状態になる場合もあります。 **LOAD QUERY** によって戻される状態は以下のとおりです。

正常状態

「正常」という状態は、表が作成された後の最初の状態であり、現在、表に表れるような (異常な) 状態がないことを示しています。

読み取りアクセスのみ

ALLOW READ ACCESS オプションを指定すると、表は「読み取りアクセスのみ」の状態になります。ロード・コマンドを呼び出す前に存在した表のデータが、ロード操作時に読み取り専用モードで使用可能になります。 **ALLOW READ ACCESS** オプションを指定したときに、ロード操作が失敗した場合には、ロード操作の前に表に存在していたデータが、失敗後も読み取り専用モードで引き続き使用可能になります。

ロード進行中

「ロード進行中」という表の状態は、表に進行中のロードがあることを示しています。ロード・ユーティリティーは、ロードが正常に完了すると、この過渡状態を解除します。しかし、ロード操作が失敗または中断されると、表の状態は「ロード・ペンディング」に変わります。

再配分進行中

「再配分進行中」という表の状態は、表に進行中の再配分があることを示しています。再配分ユーティリティーは、表の処理が正常に完了すると、この過渡状態を解除します。しかし、再配分操作が失敗または中断されると、表の状態は「再配分ペンディング」に変わります。

ロード・ペンディング

「ロード・ペンディング」という表の状態は、ロード操作が失敗または中断されたことを示しています。以下のいずれかのステップを行うことにより、「ロード・ペンディング」状態を解除することができます。

- 失敗の原因に対処します。例えば、ロード・ユーティリティーがディスク・スペースを使い果たした場合、表スペースにコンテナーを追加します。その後、ロード操作を再開します。
- ロード操作を終了します。
- ロード操作が失敗したその同じ表に対し、LOAD **REPLACE** 操作を実行します。
- ロードする表の表スペースを、最新の表スペースまたはデータベース・バックアップを指定した **RESTORE DATABASE** コマンドを使ってリカバリーした後、それ以降のリカバリー処理を実行します。

再配分ペンディング

「再配分ペンディング」という表の状態は、再配分操作が失敗または中断されたことを示します。 **REDISTRIBUTE CONTINUE** または **REDISTRIBUTE ABORT** 操作を実行すると、「再配分ペンディング」の状態を解除できます。

ロード再始動不可

「ロード再始動不可」の状態の表は部分的にロードされ、ロード再開操作を行うことができません。表は次の 2 つの状況で「ロード再始動不可」の状態になります。

- 正常に再開または終了できなかった失敗ロード操作の後ロールフォワード操作を実行した場合
- 表の状態が「ロード進行中」または「ロード・ペンディング」になっている間に行われたオンライン・バックアップからリストア操作を実行した場合

表は「ロード・ペンディング」状態にもなります。表の「ロード再始動不可」の状態を解除するには、**LOAD TERMINATE** または **LOAD REPLACE** コマンドを発行します。

SET INTEGRITY ペンディング
「SET INTEGRITY ペンディング」という状態は、ロードされた表に未確認の制約があることを示しています。ロード・ユーティリティーは、制約のある表でロード操作を開始する際に、表をこの状態にします。表の SET INTEGRITY ペンディング状態を解除するには、SET INTEGRITY ステートメントを使用してください。

タイプ 1 索引
「タイプ 1 索引」という状態は、表が現在タイプ 1 索引を使用していることを示します。タイプ 1 索引は、バージョン 9.7 以降ではサポートされなくなりました。タイプ 2 索引への変換は、バージョン 10 へアップグレードする前に行う必要があります。それ以外の場合、表が最初にアクセスされるときに、タイプ 1 索引が自動的にタイプ 2 索引として再作成されます。

データベースをアップグレードする前にタイプ 1 索引を変換する方法について詳しくは、『タイプ 1 索引からタイプ 2 索引への変換』のトピックを参照してください。

使用不可
リカバリー不能のロード操作からロールフォワードを実行すると、表は「使用不可」状態になります。この状態の表は使用することができません。ドロップするか、バックアップからリストアする必要があります。

複数の状態にある表の例

以下のようにして、相当量のデータを持つ入力ファイル (staffdata.del) を表 NEWSTAFF にロードするとします。

```
connect to sample;
create table newstaff like staff;
load from staffdata.del of del insert into newstaff allow read access;
connect reset;
```

さらに、別のセッションを開き、次のコマンドを発行します。

```
connect to sample;
load query table newstaff;
connect reset;
```

LOAD QUERY コマンドを実行すると、NEWSTAFF 表の状態が「読み取りアクセスのみ」と「ロード進行中」であることが分かります。

```
Tablestate:
 Load in Progress
 Read Access Only
```

ロード例外表

ロード例外表は、ロード操作時にユニーク索引規則、範囲制約、およびセキュリティー・ポリシーに違反したすべての行を 1 つにまとめたレポートです。ロード例外表は **LOAD** コマンドの FOR EXCEPTION 節を使用することによって指定します。

制約事項: 例外表には、ID 列も、他のどのタイプの生成列も入れることはできません。 1 次表内に ID 列があると、例外表内のそれに対応する列には、その列のタイプ、長さ、および NULL 可能性の属性しか入れることはできません。さらに、例外表をパーティション化したり、例外表にユニーク索引を入れたりすることもできません。さらに、以下の場合も例外表を指定できません。

- ターゲット表が LBAC セキュリティーを使用しており、少なくとも 1 つの XML 列がある。
- ターゲット表が範囲パーティション表で、少なくとも 1 つの XML 列がある。

ロード・ユーティリティーで使用される例外表は、SET INTEGRITY ステートメントが使用する例外表と同一のものです。これはロード中の表の定義を反映するユーザー作成の表であり、追加の列がいくつか入っています。

ロード例外表は、ロードされている表の存在する表スペース、またはその他の表スペースに割り当てることができます。どちらにしても、ロード例外表およびロードされている表は同じデータベース・パーティション・グループに割り当て、どちらも同じ分散キーを使用するようにしてください。また、例外表およびロードの対象表が、同じパーティション・マップ ID (SYSIBM.SYSTABLES.PMAP_ID) を持つようにしてください。ただし、この ID は、再配分操作 (データベース・パーティションの追加/ドロップ操作) 時に変更される可能性があります。

例外表を使用する状況

ユニーク索引を含むデータ、および重複レコードが含まれている可能性のあるデータをロードするときは、例外表を使用します。例外表を指定していない場合に重複レコードが検出されると、ロード操作はそのまま継続してしまい、削除された重複レコードに関する警告メッセージだけが出されます。重複レコードはログに記録されません。

ロード操作の完了後、エラーになったデータを例外表の中にある情報を使って訂正することができます。その後、訂正したデータを表に挿入できます。

行は例外表の中の既存の情報に追加されます。表が空であることを確かめるための検査は行われないため、前回のロード操作で無効だった行に新規の情報が単に追加されます。現在のロード操作で無効な行だけが必要な場合には、ユーティリティーを呼び出す前に既存の行を削除しておくことができます。または、ロード操作を定義するときに、違反が発見された時刻、および違反した制約の名前を例外表に記録するように指定できます。

削除イベントは発生するたびに記録されるため、固有性の条件に違反するレコードが多数あると、ロードの削除フェーズ中にログが満杯になる可能性があります。

無効なデータが原因で索引の構築前にリジェクトされた行は、例外表に挿入されません。

失敗した、または不完全なロード

中断したロード操作の再始動

ロード操作中に失敗や中断が発生した場合は、ロード・ユーティリティーを使用して操作を終了するか、表を再ロードするか、またはロード操作を再始動することができます。

実在しないデータ・ファイルや無効な列名などのユーザー・エラーが原因でロード・ユーティリティーを開始することすらできない場合、その操作はターゲット表を通常の状態にしたまま終了します。

ロード操作を開始すると、ターゲット表の状態は「ロード進行中」になります。操作が失敗すると表の状態は「ロード・ペンディング」に変わります。表のこの状態を解除するには、**LOAD TERMINATE** を発行して操作をロールバックするか、**LOAD REPLACE** を発行して表全体を再ロードするか、または **LOAD RESTART** を発行することができます。

一般的に、この状態における最良の選択はロード操作を再始動することです。ロード・ユーティリティーはロード操作を最初からではなく、その進行内の最後に正常に到達した点から再開するので、この選択は時間の節約になります。操作を再開する正確な場所は、元のコマンドで指定されたパラメーターによって異なります。SAVECOUNT オプションが指定されており、前回のロード操作が失敗した場所がロード・フェーズである場合、ロード操作は到達した最後の整合点から再開します。上記以外の場合、ロード操作は正常に到達した最後のフェーズ (ロード、構築、または削除フェーズ) の最初から再開します。

XML 文書をロードしている場合は、動作が若干異なります。SAVECOUNT オプションは XML データのロードではサポートされていないため、ロード・フェーズ中に失敗したロード操作は、操作の最初から再開します。他のデータ・タイプと同様に、構築フェーズ中にロードが失敗した場合、REBUILD モードでは索引が構築されるので、各行からすべての索引キーを取り出すために表がスキャンされます。しかし、各 XML 文書も索引キーを選択するためにスキャンされる必要があります。キーのために XML 文書をスキャンするこの処理には、文書を再び構文解析することが必要ですが、これはコストのかかる操作になります。さらに、内部の XML 索引 (領域の索引やパスの索引など) が、最初に再構築される必要があります。これには、XDA オブジェクトのスキャンも必要です。

ロード操作の失敗の原因となった状況を修正した後、ロード・コマンドを再発行します。元のコマンドと同じパラメーターを正確に指定することにより、必要な一時ファイルをロード・ユーティリティーが見つけられるようにしてください。この例外は、読み取りアクセスを許可しない場合です。 ALLOW READ ACCESS オプションを指定したロード操作は、ALLOW NO ACCESS オプションとして再開することも可能です。

注: ロード・ユーティリティーが作成した一時ファイルは、決して削除したり変更したりしないでください。

以下のコマンドの実行によりロード操作が発生し、そのロード操作が失敗したとします。

```
LOAD FROM file_name OF file_type
SAVECOUNT n
MESSAGES message_file
load_method
INTO target_tablename
```

この場合、指定したロード・メソッド (load_method) を RESTART メソッドで置き換えることによってこれを再開します。

```
LOAD FROM file_name OF file_type
SAVECOUNT n
MESSAGES message_file
RESTART
INTO target_tablename
```

失敗したロードのうち、再開できないもの

操作に関連した表の状態が「ロード再始動不可」になっている場合は、失敗または中断されたロード操作を再開することができません。表がその状態になるのには、以下の理由があります。

- 正常に再開または終了されなかった失敗ロード操作の後ロールフォワード操作が実行された
- 表の状態が「ロード進行中」または「ロード・ペンディング」になっている間に行われたオンライン・バックアップからリストア操作が実行された

LOAD TERMINATE または **LOAD REPLACE** コマンドのいずれかを発行してください。

失敗したロードの制約

BACKUP DATABASE コマンドは、**LOAD** コマンドが SMS 表スペースの表で失敗し、その表の状態が「ロード・ペンディング」のままになると、入出力エラーを戻すことがあります。

表が「ロード・ペンディング」状態の場合、表データは整合していないことがあります。表データが不整合になっていると、**BACKUP DATABASE** コマンドが失敗します。その表は、その後に **LOAD TERMINATE**、**LOAD RESTART**、**LOAD REPLACE** のいずれかのコマンドが完了するまでは不整合なままになります。

データベースをバックアップする前に、表が「ロード・ペンディング」でない状態にする必要があります。

ALLOW READ ACCESS ロード操作の再開または終了

ALLOW READ ACCESS パラメーターを指定するロード操作が中断またはキャンセルされた場合にも、**ALLOW READ ACCESS** パラメーターを使用してその操作を再開または終了することができます。 **ALLOW READ ACCESS** パラメーターを使用することにより、終了または再開操作の進行中に、他のアプリケーションは表データを照会することができます。 **ALLOW READ ACCESS** モードのロード操作と同様に、表はデータがコミットされるまで排他的にロックされます。

このタスクについて

索引オブジェクトが使用できない場合や、無効のマークが付いている場合には、**ALLOW READ ACCESS** モードでロード再開または終了操作を実行することは許可されません。

索引コピー・フェーズで元のロード操作が中断またはキャンセルされた場合、索引が壊れている可能性があるために、 **ALLOW READ ACCESS** モードの再開操作は許可されません。

ロード・フェーズで **ALLOW READ ACCESS** モードのロード操作が中断またはキャンセルされた場合、ロード・フェーズで再開します。ロード・フェーズ以外のフェーズで中断またはキャンセルされた場合には、構築フェーズで再開します。元のロード操作が **ALLOW NO ACCESS** モードである場合、元のロード操作が削除フェーズに到達するときに索引が有効であれば、そのフェーズで再開操作が行われます。索引に無効のマークが付いている場合には、ロード・ユーティリティーは構築フェーズからロード操作を再開します。

注: INDEXING MODE INCREMENTAL パラメーターが指定されている場合でも、すべてのロード再開操作は **REBUILD** 索引付けモードを選択します。

LOAD TERMINATE コマンドを発行すると、通常、中断またはキャンセルされたロード操作が最小限の遅延でロールバックされます。ただし、**ALLOW READ ACCESS** および **INDEXING MODE INCREMENTAL** が指定されているロード操作で **LOAD TERMINATE** コマンドを発行すると、ロード・ユーティリティーが索引をスキャンし、矛盾があればそれを修正する際に、遅延が生じる可能性があります。この遅延の長さは、索引のサイズによって異なり、ロード終了操作に **ALLOW READ ACCESS** パラメーターが指定されているかどうかにかかわりなく発生します。構築フェーズの前に元のロード操作が失敗した場合には、この遅延は発生しません。

注: 索引での矛盾を修正することから発生する遅延は、索引に無効というマークを付けて、これらを再構築することによって発生する遅延よりもはるかに小さくなります。

ロード再始動操作は、ロード再始動不可の状態にある表には実行できません。ロールフォワード操作時には、この表はロード再始動不可の表状態になる可能性があります。これはロード操作の終了前のある時点までロールフォワードを実行する場合、あるいは中断またはキャンセルされたロード操作を介してロールフォワードを実行するものの、ロード終了操作またはロード再開操作の終わりまでロールフォワードしない場合に発生します。

重要: バージョン 10.1 フィックスパック 1 以降、ALLOW READ ACCESS パラメーターは非推奨となっており、将来のリリースで除去される可能性があります。詳しくは、「*DB2 バージョン 10.1 の新機能*」の『LOAD コマンドの ALLOW READ ACCESS パラメーターが非推奨になった』を参照してください。

ロード・コピー・ロケーション・ファイルを使ったデータのリカバリー

DB2LOADREC レジストリー変数は、ロード・コピーのロケーション情報の入っているファイルを示すのに使用します。このファイルは、ロールフォワード・リカバリー中にロード・コピーの位置情報として使用されます。

DB2LOADREC には以下に関する情報があります。

- メディアの種類
- 使用するメディア装置の数

- 表のロード操作中に生成されるロード・コピーのロケーション
- ロード・コピーのファイル名 (もしあれば)

ロケーション・ファイルが存在しない場合、あるいはファイル内に一致する項目がない場合は、ログ・レコードからの情報が使用されます。

ファイル内の情報は、ロールフォワード・リカバリーの実行前に上書きされることがあります。

注:

1. 複数パーティション・データベースでは、 **db2set** コマンドを使用して、すべてのデータベース・パーティション・サーバーに **DB2LOADREC** レジストリー変数を設定する必要があります。
2. 複数パーティション・データベースでは、それぞれのデータベース・パーティション・サーバーにロード・コピー・ファイルが存在しなければならず、ファイル名 (パスも含む) は同じでなければなりません。
3. **DB2LOADREC** レジストリー変数によって識別されるファイルの項目が有効ではない場合、無効な項目を置き換える情報を提供するために古いロード・コピー・ロケーション・ファイルが使用されます。

ロケーション・ファイルには、以下の情報が提供されます。最初の 5 つのパラメーターには有効な値を指定する必要があり、これらのパラメーターはロード・コピーを識別するために使用されます。全体の構造は記録されるロード・コピーごとに繰り返されます。例えば、以下のようにします。

```
TIMestamp      19950725182542         * Time stamp generated at load time
DBPartition    0                      * DB Partition number (OPTIONAL)
SCHema         PAYROLL                * Schema of table loaded
TABlename      EMPLOYEES              * Table name
DATabasename   DBT                    * Database name
DB2instance    toronto                * DB2INSTANCE
BUFfernumber   NULL                   * Number of buffers to be used for
                                        recovery
SESsionnumber  NULL                   * Number of sessions to be used for
                                        recovery
TYPeofmedia    L                      * Type of media - L for local device
                                                       A for TSM
                                                       O for other vendors
LOCationnumber 3                      * Number of locations
   ENTry       /u/toronto/dbt.payroll.employes.001
   ENT         /u/toronto/dbt.payroll.employes.002
   ENT         /dev/rmt0
TIM            19950725192054
DBP            18
SCH            PAYROLL
TAB            DEPT
DAT            DBT
DB2            toronto
BUF            NULL
SES            NULL
TYP            A
TIM            19940325192054
SCH            PAYROLL
TAB            DEPT
DAT            DBT
DB2            toronto
```

```
BUF             NULL
SES             NULL
TYP             O
SHRlib          /@sys/lib/backup_vendor.a
```

注:

1. 重要なのは各キーワードの最初の 3 文字です。すべてのキーワードは、指定された順序になっている必要があります。ブランク行は使用できません。
2. タイム・スタンプのフォーマットは *yyyymmddhhmmss* です。
3. フィールドはすべて必須ですが、BUF と SES (NULL 可能)、および DBP (リストから省略可能) は例外です。 SES が NULL の場合には、 *dft_loadrec_ses* 構成パラメーターによって指定される値が使用されます。 BUF が NULL の場合には、デフォルト値は SES+2 です。
4. ロケーション・ファイルにある項目が 1 つでも無効な場合には、以前のロード・コピー・ロケーション・ファイルの値が使用されます。
5. メディアの種類には、ローカル装置 (テープ、ディスク、またはディスケットの場合は L)、 TSM (A)、あるいは他のベンダー (O) のいずれかを指定できます。種類が L の場合、ロケーション項目に続けてロケーションの数を指定する必要があります。種類が A の場合、それ以上入力する必要はありません。種類が O の場合は、共有ライブラリー名を指定する必要があります。
6. SHRlib パラメーターは、ロード・コピー・データを格納する機能を持つライブラリーを指します。
7. COPY NO または NONRECOVERABLE オプションを指定してロード操作を呼び出し、かつ操作の完了後にデータベースまたは関連する表スペースのバックアップ・コピーを取らない場合、ロード操作の後の時点でこのデータベースまたは表スペースをリストアすることはできません。つまり、ロールフォワード・リカバリーを使ってもデータベースや表スペースをロード操作後の状態に再作成することはできません。データベースや表スペースをリストアできるのは、ロード操作より前の時点の状態だけです。

特定のロード・コピーを使用する場合には、データベースのリカバリー履歴ファイルを使用して、特定のロード操作のタイム・スタンプを判別できます。複数パーティション・データベースでは、リカバリー履歴ファイルは各データベース・パーティションにローカルに存在します。

ロード・ダンプ・ファイル

dumpfile ファイル・タイプ修飾子を指定すると、リジェクトされた行を書き込む例外表の名前とロケーションをロード・ユーティリティーに対して指示できます。

パーティション・データベース環境での実行時には、パーティション化サブエージェントまたはロード・サブエージェントによって行がリジェクトされることがあります。そのため、ダンプ・ファイル名には、サブエージェントのタイプを特定する拡張子と、例外が生成されたデータベース・パーティション番号が付けられます。例えば、次のようなダンプ・ファイル値を指定したとします。

```
dumpfile = "/u/usrname/dumpit"
```

この場合、データベース・パーティション 5 でロード・サブエージェントによってリジェクトされた行は、 /u/usrname/dumpit.load.005 という名前のファイルに保

管され、データベース・パーティション 2 でロード・サブエージェントによってリジェクトされた行は、 /u/usrname/dumpit.load.002 という名前のファイルに保管され、データベース・パーティション 2 でパーティション化サブエージェントによってリジェクトされた行は、 /u/usrname/dumpit.part.002 という名前のファイルに保管される、などとなります。

ロード・サブエージェントによって行がリジェクトされた場合に、その行が 32,768 バイト未満の長さであると、そのレコード全体がダンプ・ファイルにコピーされますが、それ以上の長さの場合は、行の断片 (レコードの最終バイトを含む) がファイルに書き込まれます。

パーティション化サブエージェントによって行がリジェクトされた場合、レコード・サイズに関係なく、その行全体がダンプ・ファイルにコピーされます。

ロード一時ファイル

DB2 データベース・システムは、ロード処理中にバイナリーの一時ファイルを作成します。 このファイルは、ロード・クラッシュ・リカバリー、ロード終了操作、警告およびエラー・メッセージ、および実行時制御データに使用されます。

ロード一時ファイルは、ロード操作がエラーなしで完了した時点で削除されます。一時ファイルが書き込まれるパスは、 **LOAD** コマンドの *temp-pathname* パラメーターまたは db2Load API の *piTempFilesPath* パラメーターで指定できます。デフォルトのパスは、データベース・ディレクトリーのサブディレクトリーです。

一時ファイルのパスはサーバー・マシン上にあり、DB2 のインスタンスが排他的にアクセスします。そのため、*temp-pathname* パラメーターに指定するパス名の修飾はクライアントではなくサーバーのディレクトリー構造を反映したものでなければならず、 DB2 インスタンス所有者にはそのパスに対して読み取りと書き込みの両方の許可が必要です。

注: DB2 pureScale® 環境の場合、ロード一時ファイルは、すべてのメンバーからアクセス可能なパス上 (共有ディスクなど) に存在している必要があります。共有ディスク上に必要な一時ファイルが存在しないと、別のメンバーから実行されたメンバー・クラッシュ・リカバリーおよび **LOAD TERMINATE** 操作に、問題が生じる可能性があります。

これは、パーティション・データベース環境の場合とは異なります。パーティション・データベース環境では、ロード一時ファイルのパスはローカル・ディスク上に存在する必要があります。ネットワーク・ファイル・システム (NFS) をベースとするパスを選択するのは避けるべきです。このようなパスを選択すると、ロード操作のパフォーマンスが大幅に低下します。

重要: このパスに書き込まれる一時ファイルは、どのような状況にあっても決して手を加えないでください。一時ファイルに手を加えるとロード操作における誤動作の原因となり、データベースが危険な状態になります。

ロード・ユーティリティーのログ・レコード

ユーティリティー管理機能は、ロード・ユーティリティーなどのいくつかの DB2 ユーティリティーに関連するログ・レコードを生成します。

以下のログ・レコードには、ロード操作時の特定の活動の開始点または終了点が記録されます。

- **セットアップ・フェーズ**
  - ロード開始。このログ・レコードは、ロード操作のセットアップ・フェーズの開始を示します。
  - コミット・ログ・レコード。このログ・レコードは、セットアップ・フェーズが正常に完了したことを示します。
  - アボート・ログ・レコード。このログ・レコードは、セットアップ・フェーズが失敗したことを示します。(また単一パーティション・データベースで、表に物理的な変更が加えられる前に、ロード・セットアップ・フェーズで障害が起こる場合、ローカル・ペンディング・コミット・ログ・レコードが生成されます)。
- **ロード・フェーズ**
  - ロード開始。このログ・レコードは、ロード操作のロード・フェーズの開始を示します。
  - ローカル・ペンディング・コミット・ログ・レコード。このログ・レコードは、ロード・フェーズが正常に完了したことを示します。
  - アボート・ログ・レコード。このログ・レコードは、ロード・フェーズが失敗したことを示します。
- **削除フェーズ**
  - ロード削除開始。このログ・レコードはロード操作の削除フェーズの開始に関連したものです。削除フェーズは、主キー値が重複している場合にのみ開始されます。削除フェーズでは、表レコードに対する各削除操作または索引キーが記録されます。
  - ロード削除終了。このログ・レコードはロード操作の削除フェーズの終了に関連したものです。この削除フェーズは、正常なロード操作のロールフォワード・リカバリー中に繰り返されます。

以下のリストは、ロード・ユーティリティーが、入力データのサイズに従って作成するログ・レコードを説明します。

- ユーティリティーによって、DMS 表スペースに割り振られたり削除されたりするすべての表スペース・エクステントにつき、2 つのログ・レコードが作成されます。
- 消費される ID 値の塊ごとにログ・レコードが作成されます。
- ログ・レコードは、ロード操作の削除フェーズで削除されるデータ行または索引キーごとに作成されます。
- ALLOW READ ACCESS および INDEXING MODE INCREMENTAL オプションを指定してロード操作を実行する際に、索引ツリーの整合性を保守するためのログ・レコードが作成されます。ログ記録されるレコードの数は、完全にログ記録される索引への挿入よりずっと少なくなります。

ロードの概要 - パーティション・データベース環境

複数パーティション・データベースでは、大量のデータが多数のデータベース・パーティションに散在しています。データの各部分がどのデータベース・パーティシ

ョンに入るかは、分散キーによって決定されます。また、適切なデータベース・パーティションにデータをロードする前に、データを分散 しておく必要があります。

複数パーティション・データベースに表をロードする際に、ロード・ユーティリティーは以下を実行できます。

- 入力データを並列で分散します。
- データをそれぞれ対応するデータベース・パーティションに同時にロードします。
- あるシステムから別のシステムにデータを転送します。

複数パーティション・データベースへのデータのロードは、セットアップとロードの 2 つのフェーズで実行されます。セットアップ・フェーズ では表ロックなどのデータベース・パーティション・リソースが獲得され、ロード・フェーズ ではデータがデータベース・パーティションにロードされます。 **LOAD** コマンドの ISOLATE_PART_ERRS オプションを使用すると、これらのフェーズのいずれかで発生するエラーを処理する方法、および 1 つまたは複数のデータベース・パーティションでのエラーが、エラーのないデータベース・パーティションでのロード操作にどのように影響を与えるかを選択できます。

複数パーティション・データベースにデータをロードする際には、以下のいずれかのモードを使用できます。

PARTITION_AND_LOAD
データは (多くの場合は並列で) 分散され、それぞれ対応するデータベース・パーティションに同時にロードされます。

PARTITION_ONLY
データは (多くの場合は並列で) 分散され、それぞれのロード・データベース・パーティションの指定したファイルに出力が書き込まれます。それぞれのファイルにはパーティション・ヘッダーがあり、データがいくつかのデータベース・パーティションに分散された方法、および LOAD_ONLY モードを使用してデータベースにファイルをロードできることを示します。

LOAD_ONLY
データはすでにいくつかのデータベース・パーティションに分散されているものとします。この場合は分散プロセスが省略され、データはそれぞれ対応するデータベース・パーティションに同時にロードされます。

LOAD_ONLY_VERIFY_PART
データはすでにいくつかのデータベース・パーティションに分散されているものとしますが、データ・ファイルにはパーティション・ヘッダーがありません。分散プロセスは省略され、データはそれぞれ対応するデータベース・パーティションに同時にロードされます。ロード操作時には、それぞれの行が正しいデータベース・パーティションにあることがチェックされます。dumpfile ファイル・タイプ修飾子が指定されている場合には、データベース・パーティション違反のある行がダンプ・ファイルに入れられます。そうでなければ、行は廃棄されます。ロードしている特定のデータベース・パーティションにデータベース・パーティション違反がある場合、そのデータベース・パーティションのロード・メッセージ・ファイルに 1 つの警告が書き込まれます。

ANALYZE
すべてのデータベース・パーティションに均一に分散する最適な分散マップが生成されます。

概念および用語

複数データベース・パーティションを持つパーティション・データベース環境でのロード・ユーティリティーの動作および操作について説明する際には、以下の用語が使用されます。

- *コーディネーター・パーティション* は、ロード操作を実行するためにユーザーが接続するデータベース・パーティションです。
  PARTITION_AND_LOAD、PARTITION_ONLY、および ANALYZE モードでは、**LOAD** コマンドの CLIENT オプションが指定されていない限り、データ・ファイルはこのデータベース・パーティションに存在するものとされます。 CLIENT を指定すると、ロードされるデータが、リモートで接続されるクライアントに存在することを示します。
- PARTITION_AND_LOAD、PARTITION_ONLY、および ANALYZE モードでは、*事前パーティション化エージェント* がユーザー・データを読み取り、データを分散するパーティション化エージェント にラウンドロビン方式でこれを分散します。この処理は、コーディネーター・パーティションで常に実行されます。どのロード操作の場合でも、 1 つのデータベース・パーティションにつき最大 1 つのパーティション化エージェントが許可されます。
- PARTITION_AND_LOAD、LOAD_ONLY、および LOAD_ONLY_VERIFY_PART モードでは、それぞれの出力データベース・パーティションで*ロード・エージェント* が実行し、そのデータベース・パーティションへのデータのロードを調整します。
- *ファイル・エージェントへのロード* は、 PARTITION_ONLY ロード操作時にそれぞれの出力データベース・パーティションで実行します。これらはパーティション化エージェントからデータを受信し、これをデータベース・パーティションにあるファイルに書き込みます。
- SOURCEUSEREXIT オプションを使用すると、カスタマイズしたスクリプトまたは実行ファイル (ここでは*ユーザー出口*と呼びます) をロード・ユーティリティーが実行するための機構が提供されます。

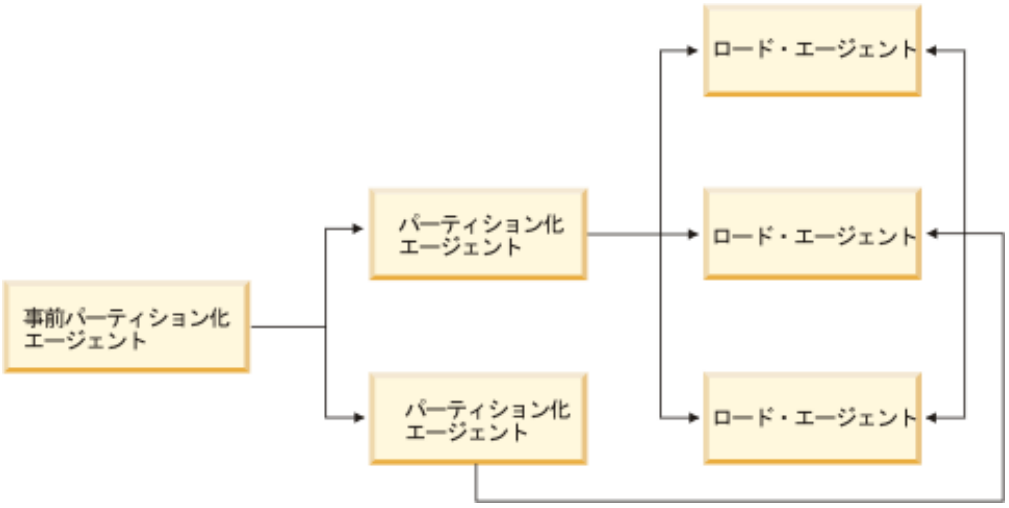


図 13. パーティション・データベース・ロードの概説： 事前パーティション化エージェントによりソース・データを読み、2 つのパーティション化エージェントそれぞれに約半分のデータが送られます。パーティション化エージェントはデータを分散し、各パーティションを 3 つのデータベース・パーティションのいずれかに送ります。各データベース・パーティションのロード・エージェントがデータをロードします。

パーティション・データベース環境でのデータのロード

ロード・ユーティリティーを使用して、パーティション・データベース環境にデータをロードします。

始める前に

複数パーティションを持つデータベースに表をロードする前に確認する事項：

- データベース・マネージャー構成パラメーター **svcename**、およびプロファイル・レジストリー変数 **DB2COMM** が正しく設定されていることを確認してください。ロード・ユーティリティーは TCP/IP を使用して事前パーティション化エージェントからパーティション化エージェントにデータを転送し、さらにパーティション化エージェントからロード・データベース・パーティションにデータを転送するため、このステップは重要です。
- ロード・ユーティリティーを起動するには、その前にデータのロード先となるデータベースに接続されているか、または暗黙接続が可能な状態になっていなければなりません。
- ロード・ユーティリティーは COMMIT ステートメントを発行するので、ロード操作を開始する前に COMMIT または ROLLBACK ステートメントを発行して、すべてのトランザクションを完了し、すべてのロックを解除しておいてください。 PARTITION_AND_LOAD、 PARTITION_ONLY、または ANALYZE モードが使用されている場合には、ロードされるデータ・ファイルは、以下の状態になっていない限り、このデータベース・パーティションになければなりません。
  1. **CLIENT** パラメーターが指定されている場合。この場合には、データがクライアント・マシンになければなりません。
  2. 入力ソース・データが CURSOR である場合。この場合には、入力ファイルはありません。

- 設計アドバイザーを実行して、各表ごとに最適なデータベース・パーティションを決定します。詳しくは、「*問題判別およびデータベース・パフォーマンスのチューニング*」の『設計アドバイザー』を参照してください。

制約事項

ロード・ユーティリティーを使用して複数パーティション・データベースにデータをロードする際には、以下の制約が適用されます。

- ロード操作の入力ファイルのロケーションとして磁気テープ装置を指定することはできません。
- **ANALYZE** モードが使用されていない限り、ROWCOUNT パラメーターはサポートされません。
- ターゲット表に分散に必要な ID 列があり、**identityoverride** ファイル・タイプ修飾子が指定されていない場合、または複数のデータベース・パーティションを使ってデータを分散してからロードする場合、**LOAD** コマンドにおいて 0 より大きい **SAVECOUNT** 値の使用はサポートされていません。
- ID 列が分散キーの一部を構成している場合は、PARTITION_AND_LOAD モードだけがサポートされます。
- **LOAD** コマンドの CLIENT パラメーターを指定する際には、LOAD_ONLY および **LOAD_ONLY_VERIFY_PART** モードは使用できません。
- CURSOR 入力ソース・タイプを指定する際には、LOAD_ONLY_VERIFY_PART モードは使用できません。
- **LOAD** コマンドの ALLOW READ ACCESS および COPY YES パラメーターを指定する際には、分散エラー分離モード **LOAD_ERRS_ONLY** および **SETUP_AND_LOAD_ERRS** は使用できません。
- **OUTPUT_DBPARTNUMS** および **PARTITIONING_DBPARTNUMS** オプションによって指定されるデータベース・パーティションがオーバーラップしない場合には、複数のロード操作が同じ表に同時にデータをロードすることができます。例えば、表がデータベース・パーティション 0 から 3 に定義されている場合、 1 つのロード操作がデータベース・パーティション 0 および 1 にデータをロードする一方で、2 番目のロード操作でデータベース・パーティション 2 および 3 にデータをロードすることができます。 **PARTITIONING_DBPARTNUMS** オプションによって指定されたデータベース・パーティションがオーバーラップしている場合、対象の表でロード・パーティション化サブエージェントがまだ実行されていないとロード操作で自動的に **PARTITIONING_DBPARTNUMS** パラメーターが選択され、使用可能なデータベース・パーティションがない場合には LOAD が失敗します。

  バージョン 9.7 フィックスパック 6 以降では、**PARTITIONING_DBPARTNUMS** オプションによって指定されたデータベース・パーティションがオーバーラップしている場合、表でロード・パーティション化サブエージェントがまだ実行されていないと、ロード・ユーティリティーは、**OUTPUT_DBPARTNUMS** で示されているデータベース・パーティションから自動的に **PARTITIONING_DBPARTNUMS** パラメーターの選択を試みます。使用可能なデータベース・パーティションがない場合には失敗します。

  PARTITIONING_DBPARTNUMS オプションを使用してパーティションを明示的に指定する予定の場合は、そのオプションをすべての並行 **LOAD** コマンドで使用し、各

コマンドで別々のパーティションを指定することを強くお勧めします。並行ロード・コマンドの一部のみで **PARTITIONING_DBPARTNUMS** を指定した場合、または、オーバーラップするパーティションを指定した場合は、 **LOAD** コマンドは並行ロードの少なくとも一部で代替パーティション化ノードを選択しなければならず、その場合まれなケースとしてコマンドが失敗することがあります (SQL2038N)。

- 区切りなし ASCII (ASC) および区切り付き ASCII (DEL) ファイルのみ、複数のデータベース・パーティションにまたがる複数の表に分散できます。PC/IXF ファイルは分散できませんが、LOAD_ONLY_VERIFY_PART モードでロード操作を行って、複数のデータベース・パーティションに分散されている表にロードすることはできます。

例

以下の例では、**LOAD** コマンドを使用して各種のロード操作を開始する方法を説明します。以下の例で使用されるデータベースには、5 つのデータベース・パーティション、0、1、2、3、および 4 があります。データベース・パーティションにはそれぞれローカル・ディレクトリー /db2/data/ があります。データベース・パーティション 0、1、3、および 4 では、2 つの表、TABLE1 および TABLE2 が定義されます。クライアントからロードする際には、ユーザーにはデータベース・パーティションのいずれかではないリモート・クライアントへのアクセスがあります。

分散およびロードの例

このシナリオでは、TABLE1 が定義されているか、または定義されていないデータベース・パーティションに接続しているものとします。データ・ファイル load.del は、このデータベース・パーティションの現行作業ディレクトリーにあります。 load.del から、TABLE1 が定義されているすべてのデータベース・パーティションにデータをロードするには、次のコマンドを発行します。

```
LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1
```

注: この例では、すべてのパーティション・データベース環境の構成パラメーターでデフォルト値を使用します。**MODE** パラメーターのデフォルトは PARTITION_AND_LOAD です。 **OUTPUT_DBPARTNUMS** パラメーターのデフォルトは、TABLE1 が定義されているすべてのデータベース・パーティションです。また、**PARTITIONING_DBPARTNUMS** のデフォルトは、**LOAD** コマンド規則に従って選択したデータベース・パーティションのセットです。この規則は、データベース・パーティションが指定されていない場合にそれを選択するためのものです。

データがデータベース・パーティション 3 および 4 に分散されるようにロード操作を実行するには、以下のコマンドを発行します。

```
LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1
 PARTITIONED DB CONFIG PARTITIONING_DBPARTNUMS (3,4)
```

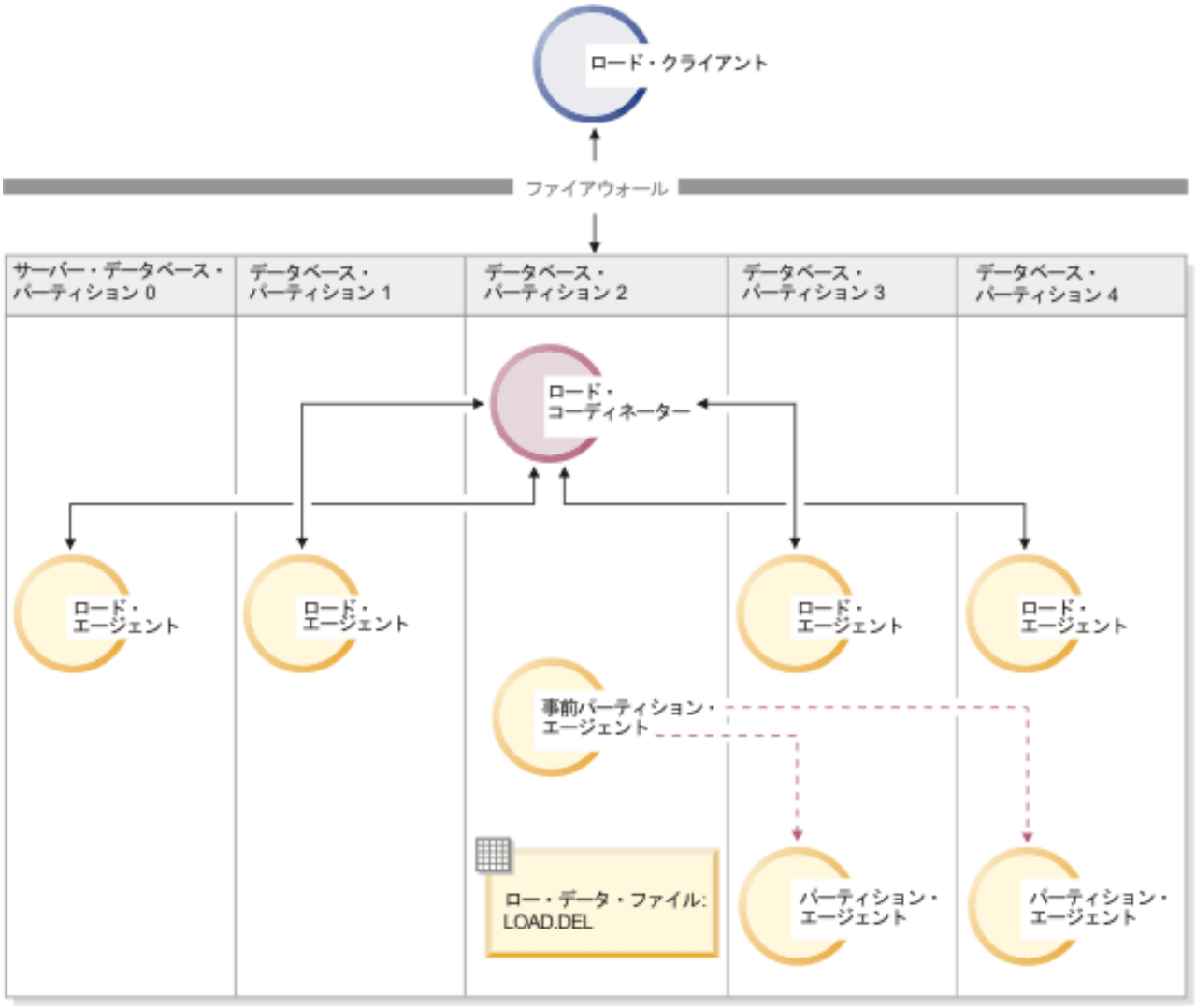


図 14. データベース・パーティション 3 および 4 へのデータのロード。 : この図では、前のコマンドが発行された結果の動作を説明します。データは、データベース・パーティション 3 および 4 にロードされます。

分散のみの例

このシナリオでは、TABLE1 が定義されているか、または定義されていないデータベース・パーティションに接続しているものとします。データ・ファイル load.del は、このデータベース・パーティションの現行作業ディレクトリーにあります。 TABLE1 が定義されているすべてのデータベース・パーティションに load.del を分散する (ロードはしない) には、データベース・パーティション 3 および 4 を使用し、以下のコマンドを発行します。

```
LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1
 PARTITIONED DB CONFIG MODE PARTITION_ONLY
 PART_FILE_LOCATION /db2/data
 PARTITIONING_DBPARTNUMS (3,4)
```

これにより、ファイル load.del.*xxx* は、それぞれのデータベース・パーティションにある /db2/data ディレクトリーに保管されます。ここで、*xxx* は、3 桁表記のデータベース・パーティション番号です。

データベース・パーティション 0 (**PARTITIONING_DBPARTNUMS** のデフォルト) で実行しているパーティション化エージェントを 1 つだけ使用して、load.del ファイルをデータベース・パーティション 1 および 3 に分散するには、以下のコマンドを発行します。

```
LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
 PARTITIONED DB CONFIG MODE PARTITION_ONLY
 PART_FILE_LOCATION /db2/data
 OUTPUT_DBPARTNUMS (1,3)
```

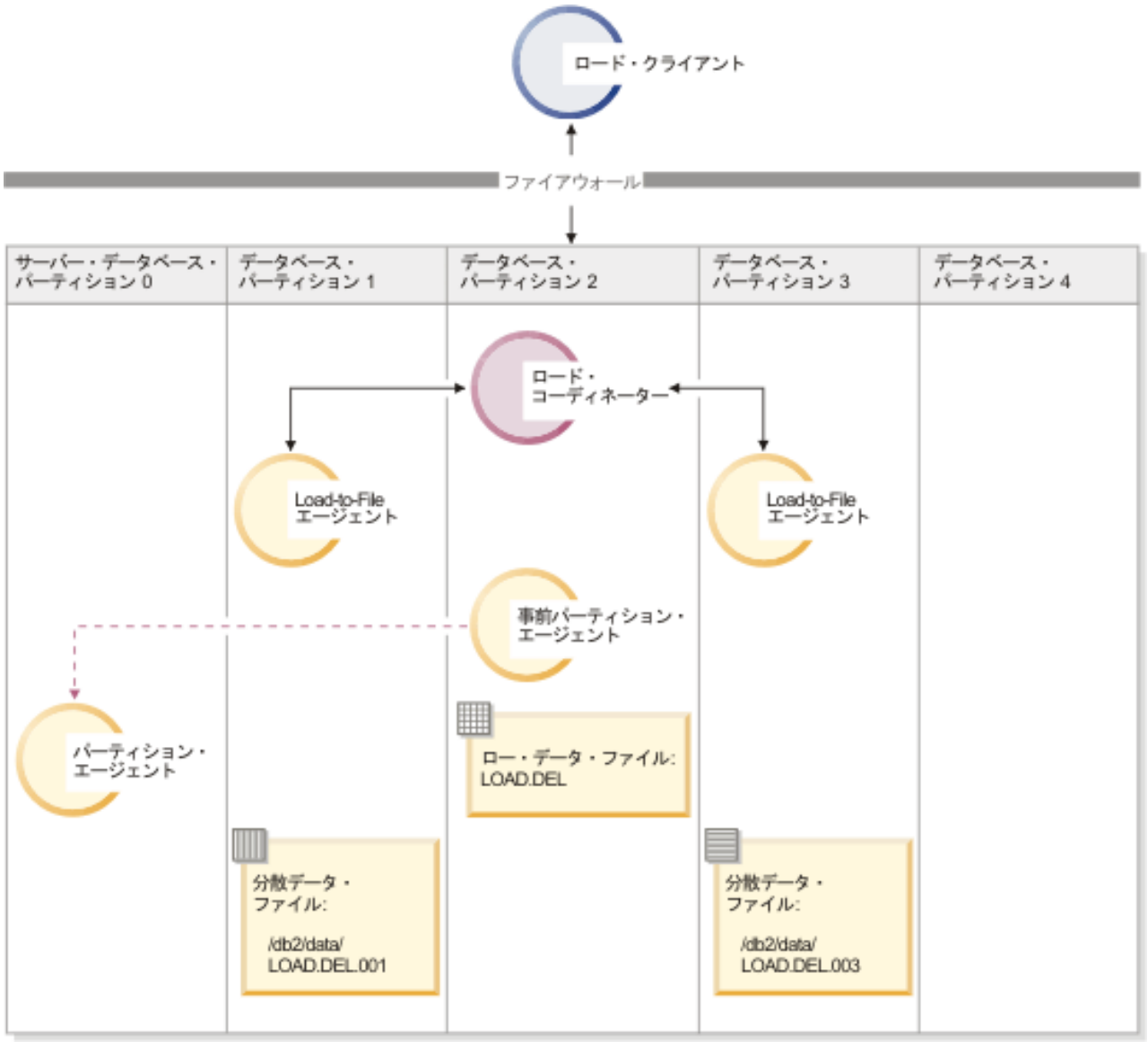


図 15. 1 つのパーティション化エージェントを使用したデータベース・パーティション 1 および 3 へのデータのロード。: この図では、前のコマンドが発行された結果の動作を説明します。データは、データベース・パーティション 0 で実行する 1 パーティション化エージェントを使用して、データベース・パーティション 1 および 3 にロードされます。

ロードのみの例

すでに PARTITION_ONLY モードでロード操作を実行しており、 TABLE1 が定義されているすべてのデータベース・パーティションにそれぞれのロード・データベース・パーティションの /db2/data ディレクトリーにあるパーティション・ファイルをロードする場合には、以下のコマンドを発行します。

```
LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
 PARTITIONED DB CONFIG MODE LOAD_ONLY
 PART_FILE_LOCATION /db2/data
```

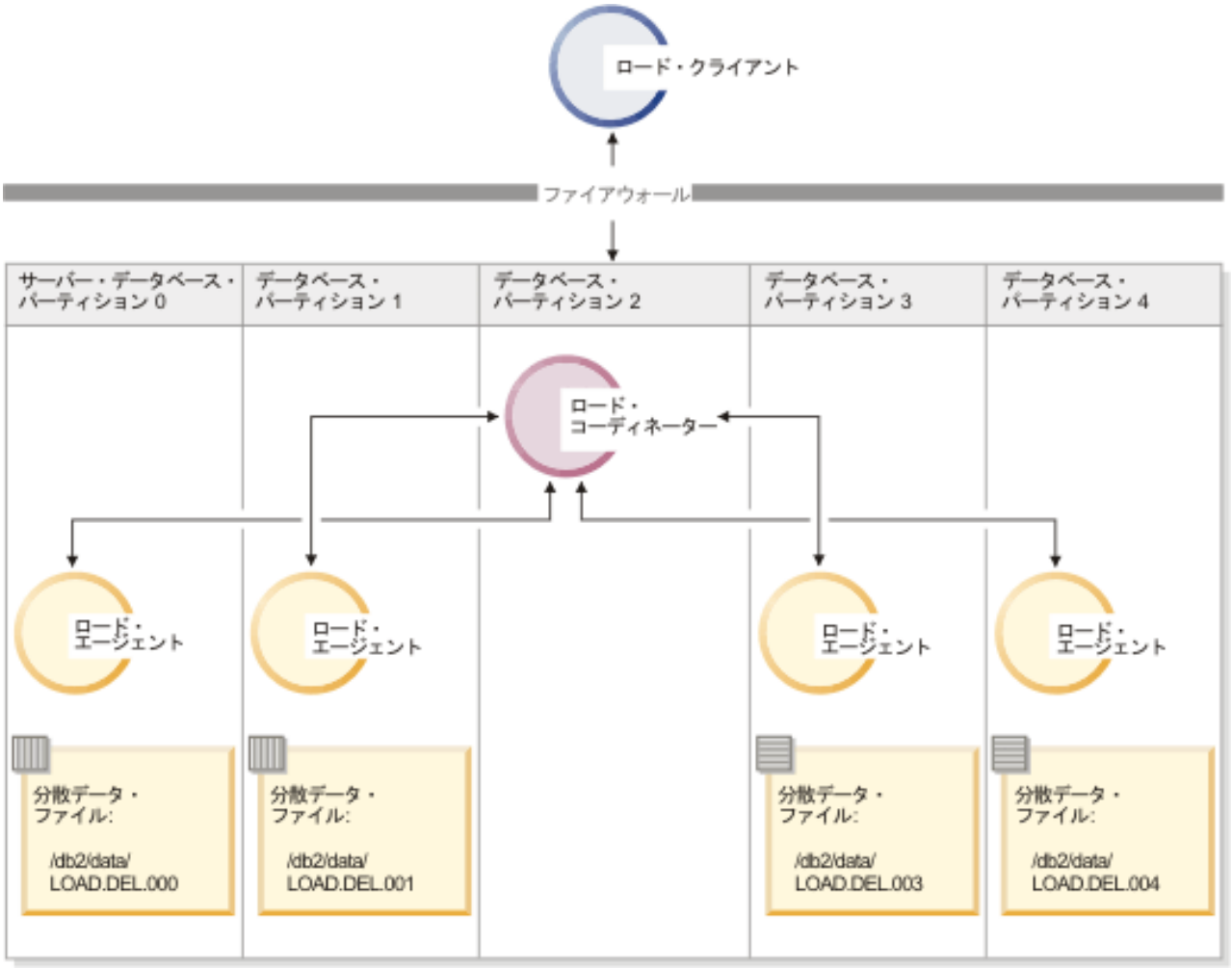


図 16. 特定の表が定義されているすべてのデータベース・パーティションへのデータのロード。 : この図では、前のコマンドが発行された結果の動作を説明します。分散データは、TABLE1 が定義されているすべてのデータベース・パーティションにロードされます。

データベース・パーティション 4 にだけロードするには、以下のコマンドを発行します。

```
LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
 PARTITIONED DB CONFIG MODE LOAD_ONLY
 PART_FILE_LOCATION /db2/data
 OUTPUT_DBPARTNUMS (4)
```

分散マップ・ヘッダーのない事前分散ファイルのロード

LOAD コマンドを使用して、分散ヘッダーのないデータ・ファイルを、いくつかのデータベース・パーティションに直接ロードすることができます。TABLE1 が定義されているそれぞれのデータベース・パーティションの /db2/data ディレクトリーにデータ・ファイルが存在しており、この名前が load.del.*xxx* である場合 (*xxx* はデータベース・パーティション番号)、以下のコマンドを発行することによってファイルをロードできます。

```
LOAD FROM LOAD.DEL OF DEL modified by dumpfile=rejected.rows
 REPLACE INTO TABLE1
 PARTITIONED DB CONFIG MODE LOAD_ONLY_VERIFY_PART
 PART_FILE_LOCATION /db2/data
```

データベース・パーティション 1 にだけデータをロードするには、以下のコマンドを発行します。

```
LOAD FROM LOAD.DEL OF DEL modified by dumpfile=rejected.rows
 REPLACE INTO TABLE1
 PARTITIONED DB CONFIG MODE LOAD_ONLY_VERIFY_PART
 PART_FILE_LOCATION /db2/data
 OUTPUT_DBPARTNUMS (1)
```

注: ロード元のデータベース・パーティションにない行が指定された場合には、これはリジェクトされ、ダンプ・ファイルに入れられます。

リモート・クライアントから複数パーティション・データベースへのロード

リモート・クライアントにあるファイルから複数パーティション・データベースにデータをロードするには、**LOAD** コマンドの **CLIENT** パラメーターを指定する必要があります。このパラメーターは、サーバー・パーティションにデータ・ファイルが存在しないことを示します。例えば、以下のようにします。

```
LOAD CLIENT FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
```

注: LOAD_ONLY または LOAD_ONLY_VERIFY_PART モードを **CLIENT** パラメーターと共に使用することはできません。

カーソルからのロード

単一パーティション・データベースの場合と同様に、カーソルから複数パーティション・データベースにロードすることができます。この例では、PARTITION_ONLY および LOAD_ONLY モードの場合、**PART_FILE_LOCATION** パラメーターは完全修飾ファイル名を指定しなければなりません。この名前は、それぞれの出力データベース・パーティションで作成またはロードされる分散ファイルの完全修飾基本ファイル名になります。ターゲット表に LOB 列がある場合には、指定されたベース名で複数のファイルを作成できます。

将来 TABLE2 にロードする目的で、/db2/data/select.out.*xxx* (ここで、*xxx* はデータベース・パーティション番号) という名前のそれぞれのデータベース・パーティションにあるファイルに、ステートメント SELECT * FROM TABLE1 の応答セット内のすべての行を分散するには、以下のコマンドを発行します。

```
DECLARE C1 CURSOR FOR SELECT * FROM TABLE1

LOAD FROM C1 OF CURSOR REPLACE INTO TABLE2
 PARTITIONED DB CONFIG MODE PARTITION_ONLY
 PART_FILE_LOCATION /db2/data/select.out
```

上記の操作によって生成されるデータ・ファイルは、以下の **LOAD** コマンドを発行することによってロードできます。

```
LOAD FROM C1 OF CURSOR REPLACE INTO TABLE2
 PARTITIONED CB CONFIG MODE LOAD_ONLY
 PART_FILE_LOCATION /db2/data/select.out
```

パーティション・データベース環境でのデータのロード - ヒント: 複数パーティション・データベースに表をロードする前に、以下のことを考慮してください。

- ロード構成オプションに慣れるために、まず小さなデータを使ってこのユーティリティーを操作してください。
- 入力データがあらかじめソートされている場合、または特定の順序になっている場合に、ロード・プロセス中にその順序を維持するとき、分散に使用するデータベース・パーティションは 1 つだけにしてください。並列分散では、必ずしもデータを受け取ったのと同じ順序でロードするとは限りません。 **LOAD** コマンドで anyorder 修飾子を指定しないと、ロード・ユーティリティーはデフォルトで単一パーティション化エージェントを選択します。
- 複数に分割されたファイルからラージ・オブジェクト (LOB) をロードする場合 (つまりロード・ユーティリティーの lobsinfile 修飾子を使っている場合)、LOB ファイルの入っているすべてのディレクトリーは、ロード先のすべてのデータベース・パーティションから読み取り可能でなければなりません。 LOB を処理する場合、ロードのパラメーター *lob-path* は完全修飾パスでなければなりません。
- ISOLATE_PART_ERRS オプションを SETUP_ERRS_ONLY または SETUP_AND_LOAD_ERRS に設定することにより、(起動時に) ロード操作でロード・データベース・パーティションや関連する表スペースまたは表がオフラインになっていることを検出した場合でも、複数パーティション・データベースで実行されているジョブを続行させることができます。
- STATUS_INTERVAL ロード構成オプションを使用して、複数パーティション・データベースで実行されているジョブの進行をモニターします。ロード操作は、指定された時間間隔でメッセージを生成し、事前パーティション化エージェントによって読み取られたデータの量をメガバイト単位で表示します。これらのメッセージは、事前パーティション化エージェント・メッセージ・ファイルにダンプされます。ロード操作中にこのファイルの内容を表示するには、コーディネーター・パーティションに接続してから、ターゲット表に対して **LOAD QUERY** コマンドを発行します。
- (PARTITIONING_DBPARTNUMS オプションで定義される) 分散プロセスに関係しているデータベース・パーティションが、 (OUTPUT_DBPARTNUMS オプションで定義される) ロード・データベース・パーティションと異なっている場合、 CPU サイクルに対する競合が少なくなるため、パフォーマンスの改善が望めます。複数パーティション・データベースにデータをロードする際には、ロード・ユーティリティーを分散操作にもロード操作にも関係しないデータベース・パーティションで起動してください。
- **LOAD** コマンドに MESSAGES パラメーターを指定すると、事前パーティション化、パーティション化、およびロード・エージェントからのメッセージ・ファイルを保管して、ロード操作の終了時に参照できるようにします。ロード操作中にこれらのファイルの内容を表示するには、適切なデータベース・パーティションに接続してから、ターゲット表に対して **LOAD QUERY** コマンドを発行します。

- ロード・ユーティリティーは、統計情報を収集する出力データベース・パーティションを 1 つだけ選択します。そのデータベース・パーティションを指定するには、 RUN_STAT_DBPARTNUM データベース構成オプションを使用できます。
- 複数パーティション・データベースでデータをロードする場合、事前に設計アドバイザーを実行して、各表ごとに最適なパーティションを判別します。詳しくは、*問題判別*およびデータベース・パフォーマンスのチューニングの『設計アドバイザー』を参照してください。

トラブルシューティング

ロード・ユーティリティーが停止した場合には、以下を実行できます。

- STATUS_INTERVAL パラメーターを使用して、複数パーティション・データベース・ロード操作の進行をモニターすることができます。状況インターバル情報は、コーディネーター・パーティションの事前パーティション化エージェント・メッセージ・ファイルにダンプされます。
- パーティション化エージェント・メッセージ・ファイルをチェックして、それぞれのデータベース・パーティションでのパーティション化エージェント・プロセスの状況を調べます。エラーなしでロードが進行しており、TRACE オプションが設定された場合には、これらのメッセージ・ファイル内に多くのレコードに関するトレース・メッセージがあるはずです。
- ロード・メッセージ・ファイルをチェックして、ロード・エラー・メッセージがあるかどうかを調べます。

  注: これらのファイルを出力するためには、**LOAD** コマンドの MESSAGES オプションを指定しなければなりません。
- ロード・プロセスのいずれかに問題が発生したことを暗示するエラーを発見した場合、現在のロード操作を中断します。

LOAD QUERY コマンドを使用したパーティション・データベース環境でのロード操作のモニター

パーティション・データベース環境でロード操作を行う際、いくつかのロード・プロセスによって、それらが実行されるデータベース・パーティションにメッセージ・ファイルが作成されます。

これらのメッセージ・ファイルには、すべての情報、ロード操作の実行時に生成される警告およびエラー・メッセージが保管されます。ユーザーが表示できるメッセージ・ファイルを生成するロード・プロセスは、ロード・エージェント、事前パーティション化エージェント、およびパーティション化エージェントです。メッセージ・ファイルの内容は、ロード操作が終了してからでなければ使用できません。

ロード操作時に個々のデータベース・パーティションに接続し、ターゲット表に対して **LOAD QUERY** コマンドを発行できます。このコマンドが CLP から発行されると、**LOAD QUERY** コマンドで指定された表の、現在このデータベース・パーティションにあるすべてのメッセージ・ファイルの内容が表示されます。

例えば、データベース・パーティション 0 から 3 で構成されるデータベース WSDB の表、TABLE1 が定義されているとします。データベース・パーティション 0 に接続され、以下の **LOAD** コマンドを発行します。

```
load from load.del of del replace into table1 partitioned db config
    partitioning_dbpartnums (1)
```

このコマンドは、データベース・パーティション 0、1、2、および 3 で実行するロード・エージェント、データベース・パーティション 1 で実行するパーティション化エージェント、データベース・パーティション 0 で実行する事前パーティション化エージェントを組み込むロード操作を開始します。

データベース・パーティション 0 には、事前パーティション化エージェントのメッセージ・ファイルが 1 つ、およびそのデータベース・パーティションでのロード・エージェントのファイルが 1 つ配備されます。これらのファイルの内容を同時に表示するには、新しいセッションを開始し、以下のコマンドを CLP から発行します。

```
set client connect_node 0
connect to wsdb
load query table table1
```

データベース・パーティション 1 には、ロード・エージェントのファイルが 1 つ、およびパーティション化エージェントのファイルが 1 つ配備されます。これらのファイルの内容を表示するには、新しいセッションを開始し、以下のコマンドを CLP から発行します。

```
set client connect_node 1
connect to wsdb
load query table table1
```

注: STATUS_INTERVAL ロード構成オプションによって生成されるメッセージは、事前パーティション化エージェント・メッセージ・ファイルに表示されます。ロード操作時にこれらのメッセージを表示するには、コーディネーター・パーティションに接続してから、**LOAD QUERY** コマンドを発行しなければなりません。

メッセージ・ファイルの内容の保管

db2Load API を介してロード操作が開始される場合には、メッセージ・オプション (piLocalMsgFileName) を指定しなければならず、メッセージ・ファイルはサーバーからクライアントに移動して、表示できるように保管されます。

CLP から開始される複数パーティション・データベースのロード操作では、メッセージ・ファイルがコンソールに表示されたり、保持されたりすることはありません。複数パーティション・データベースのロードの完了後にこれらのファイルの内容を保管または表示するには、LOAD コマンドの MESSAGES オプションを指定しなければなりません。このオプションが使用される場合、ロード操作が完了すると、それぞれのデータベース・パーティションのメッセージ・ファイルはクライアント・マシンに転送され、MESSAGES オプションによって示されるベース名を使用してファイルに保管されます。複数パーティション・データベースのロード操作の場合の、生成されたロード・プロセスに対応するファイルの名前を次の表にリストします。

プロセス・タイプ	ファイル名
ロード・エージェント	*<message-file-name>*.LOAD.*<dbpartition-number>*

プロセス・タイプ	ファイル名
パーティション化エージェント	<*message-file-name*>.PART.<*dbpartition-number*>
事前パーティション化エージェント	<*message-file-name*>.PREP.<*dbpartition-number*>

例えば、MESSAGES オプションが /wsdb/messages/load を指定すると、データベース・パーティション 2 のロード・エージェント・メッセージ・ファイルは /wsdb/messages/load.LOAD.002 です。

注: CLP から開始した複数パーティション・データベースのロード操作には、MESSAGES オプションを使用することを強くお勧めします。

パーティション・データベース環境でのロード操作の再開、再始動、または終了

パーティション・データベース環境でロード操作が失敗した場合、その後に実行すべきステップは、失敗がいつ発生したかによって異なります。

複数パーティション・データベースでのロード・プロセスは 2 つのステージで構成されます。

1. 1 つはセットアップ・ステージ です。このステージ中に、出力データベース・パーティションに対する表ロックなどのデータベース・パーティション・レベルのリソースを取得します。

   通常、セットアップ・ステージで障害が発生した場合には、操作の再始動および終了は必要ではありません。行うべき作業は、失敗したロード操作で指定されたエラー分離モードによって異なります。

   セットアップ・ステージ・エラーを分離しないことをロード操作で指定した場合、ロード操作全体がキャンセルされ、それぞれのデータベース・パーティションの表の状態は、ロード操作以前の状態にロールバックされます。

   セットアップ・ステージ・エラーを分離することをロード操作で指定した場合、ロード操作は、セットアップ・ステージが正常に実行されたデータベース・パーティションで継続しますが、失敗したそれぞれのデータベース・パーティションにある表は、ロード操作以前の状態にロールバックされます。これは、セットアップ・ステージ中に失敗するパーティションとロード・ステージ中に失敗するパーティションがある場合、単一のロード操作が複数のステージで失敗する可能性があることを意味します。
2. もう 1 つはロード・ステージ です。このステージ中にデータがフォーマットされ、データベース・パーティション上の表にロードされます。

   複数パーティション・データベースのロード操作のロード・ステージで少なくとも 1 つのデータベース・パーティションにおいてロード操作が失敗した場合には、**LOAD RESTART** または **LOAD TERMINATE** コマンドを発行しなければなりません。これが必要になるのは、複数パーティション・データベースでのデータのロードが単一のトランザクションで実行されるためです。

ロード失敗の原因となった問題を修正できる場合は、**LOAD RESTART** を選択してください。ロードの再始動操作が開始されると、ロードが中断した時点から、すべてのデータベース・パーティションでロード操作が継続されるので、これは時間の節約になります。

表を初期ロード操作前の状態に戻す場合は、**LOAD TERMINATE** を選択してください。

ロードがいつ失敗したかの判別

パーティション環境でロード操作が失敗したらまず、どのパーティションでロード操作が失敗したか、またそれぞれどのステージで失敗したかを判別する必要があります。これは、パーティション・サマリーを調べることによって行えます。 **LOAD** コマンドが CLP から発行された場合、パーティション・サマリーはロードの最後に表示されます (次の例を参照)。 **LOAD**コマンドが db2Load API から発行された場合、パーティション・サマリーは db2PartLoadOut 構造の **poAgentInfoList** フィールドで確認できます。

ある特定のパーティションの "Agent Type" の項目が "LOAD" となっている場合、そのパーティションがロード・ステージに達していることを示しています。それ以外の場合、失敗はセットアップ・ステージ中に発生したことを示します。負の SQL コードは失敗を示します。以下は、ロード・ステージ中にパーティション 1 で失敗したロードの例です。

```
Agent Type     Node     SQL Code     Result
_____________________________________________________
LOAD           000      +00000000    Success.
_____________________________________________________
LOAD           001      -00000289    Error. May require RESTART.
_____________________________________________________
LOAD           002      +00000000    Success.
_____________________________________________________
LOAD           003      +00000000    Success.
.
.
.
```

失敗したロードの再開、再始動、または終了

セットアップ・ステージ中に失敗するのは、SETUP_ERRS_ONLY または SETUP_AND_LOAD_ERRS を指定した ISOLATE_PART_ERRS オプションを使用するロードだけです。このステージ中に少なくとも 1 つの出力データベース・パーティションで失敗したロードについては、 **LOAD REPLACE** または **LOAD INSERT** コマンドを発行することができます。 OUTPUT_DBPARTNUMS オプションを使用して、失敗が発生したデータベース・パーティションのみを指定してください。

ロード・ステージ中に少なくとも 1 つの出力データベース・パーティションで失敗するロードについては、**LOAD RESTART** または **LOAD TERMINATE** コマンドを発行してください。

セットアップ・ステージ中に少なくとも 1 つの出力データベース・パーティションで、またロード・ステージ中に少なくとも 1 つの出力データベース・パーティションで失敗するロードについては、前述したとおり、失敗したロードを再開するために 2 つのロード操作 (1 つはセットアップ・ステージでの失敗に対するもので、も

う 1 つはロード・ステージでの失敗に対するもの) を実行する必要があります。このタイプの失敗ロード操作を効率的に取り消すには、**LOAD TERMINATE** コマンドを発行します。ただし、セットアップ・ステージ中に失敗したパーティション上の表に対しては変更が行われず、ロード・ステージ中に失敗したパーティションに関してはすべての変更が取り消されたため、このコマンドを発行した後はそれに伴う対応をすべてのパーティションに対して行う必要があります。

例えば、データベース・パーティション 0 から 3 で構成されるデータベース WSDB で TABLE1 が定義されているとします。以下のコマンドが発行されます。

```
load from load.del of del insert into table1 partitioned db config
isolate_part_errs setup_and_load_errs
```

セットアップ・ステージ中に出力データベース・パーティション 1 で失敗が発生します。セットアップ・ステージ・エラーは分離されるため、ロード操作は継続します。しかし、ロード・ステージ中にパーティション 3 で失敗が発生します。ロードを操作を再開するには、以下のコマンドを発行します。

```
load from load.del of del replace into table1 partitioned db config
output_dbpartnums (1)

load from load.del of del restart into table1 partitioned db config
isolate_part_errs setup_and_load_errs
```

注: ロード再始動操作では、**LOAD RESTART** コマンドで指定されるオプションが使用されるため、元の **LOAD** コマンドで指定されるものと同一であることが重要です。

マイグレーションおよびバージョン互換性

複数パーティション・データベースにおいて DB2 Universal Database™ バージョン 8 以前のロードの動作に戻る場合は、**DB2_PARTITIONEDLOAD_DEFAULT** レジストリー変数を使用できます。

注: DB2_PARTITIONEDLOAD_DEFAULT レジストリー変数は非推奨で、将来のリリースでは除去される可能性があります。

複数パーティション・データベースにおいて DB2 UDBバージョン 8 以前の **LOAD** コマンドの動作に戻ると、パーティション・データベース構成オプションを追加で指定しなくても、有効な分散ヘッダーのあるファイルを単一データベース・パーティションにロードすることができます。これは、**DB2_PARTITIONEDLOAD_DEFAULT** の値を NO に設定することによって行えます。単一データベース・パーティションに **LOAD** コマンドを発行する既存のスクリプトを変更したくない場合、このオプションを使用することをお勧めします。例えば、4 つのデータベース・パーティションを持つデータベース・パーティション・グループに属する表のデータベース・パーティション 3 に配布ファイルをロードするには、以下のコマンドを発行します。

```
db2set DB2_PARTITIONEDLOAD_DEFAULT=NO
```

それから、DB2 コマンド行プロセッサーから以下のコマンドを発行します。

```
CONNECT RESET

SET CLIENT CONNECT_NODE 3

CONNECT TO DB MYDB

LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
```

複数パーティション・データベースでは、複数パーティション・データベース・ロード構成オプションが指定されない場合には、表が定義されているすべてのデータベース・パーティションでロード操作が実行されます。入力ファイルには分散ヘッダーは必要ではなく、 MODE オプションはデフォルトの PARTITION_AND_LOAD になります。単一データベース・パーティションをロードするには、OUTPUT_DBPARTNUMS オプションを指定しなければなりません。

リファレンス - パーティション環境でのロード

パーティション・データベース環境でのロード・セッション - CLP の例:

以下の例は、複数パーティション・データベースでのデータのロードを示しています。

データベースに 0 から 3 の番号の付いた 4 つのデータベース・パーティションがあります。データベース WSDB がすべてのデータベース・パーティションで定義されており、表 TABLE1 が、すべてのデータベース・パーティションでも定義されているデフォルト・データベース・パーティション・グループにあります。

例 1

データベース・パーティション 0 にあるユーザー・データ・ファイル load.del から TABLE1 にデータをロードするには、データベース・パーティション 0 に接続してから、以下のコマンドを発行します。

```
load from load.del of del replace into table1
```

ロード操作が正常に実行された場合、出力は以下のようになります。

```
Agent Type     Node     SQL Code     Result
_____________________________________________________
LOAD           000      +00000000    Success.
_____________________________________________________
LOAD           001      +00000000    Success.
_____________________________________________________
LOAD           002      +00000000    Success.
_____________________________________________________
LOAD           003      +00000000    Success.
_____________________________________________________
PARTITION      001      +00000000    Success.
_____________________________________________________
PRE_PARTITION  000      +00000000    Success.
_____________________________________________________
RESULTS:       4 of 4 LOADs completed successfully.
_____________________________________________________

Summary of Partitioning Agents:
Rows Read                   = 100000
Rows Rejected               = 0
Rows Partitioned            = 100000

Summary of LOAD Agents:
Number of rows read         = 100000
Number of rows skipped      = 0
Number of rows loaded       = 100000
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 100000
```

出力では、それぞれのデータベース・パーティションごとに 1 つのロード・エージェントがあり、それぞれが正常に実行されたことを示しています。また、コーディネーター・パーティションで実行する事前パーティション化エージェントが 1 つ、およびデータベース・パーティション 1 で実行するパーティション化エージェントが 1 つあったことも示しています。これらのプロセスは、通常の SQL 戻りコード 0 を戻して正常に完了しました。統計のサマリーでは、事前パーティション化エージェントは 100,000 行を読み取り、パーティション化エージェントは 100,000 行を分散し、ロード・エージェントによってロードされるすべての行の合計は、100,000 行であることを示します。

例 2

以下の例では、データは PARTITION_ONLY モードで TABLE1 にロードされます。分散出力ファイルは、ディレクトリー /db/data の出力データベース・パーティションのそれぞれに保管されます。

```
load from load.del of del replace into table1 partitioned db config mode
   partition_only part_file_location /db/data
```

LOAD コマンドからの出力は、以下のようになります。

```
Agent Type     Node     SQL Code     Result
_____________________________________________________
LOAD_TO_FILE   000      +00000000    Success.
_____________________________________________________
LOAD_TO_FILE   001      +00000000    Success.
_____________________________________________________
LOAD_TO_FILE   002      +00000000    Success.
_____________________________________________________
LOAD_TO_FILE   003      +00000000    Success.
_____________________________________________________
PARTITION      001      +00000000    Success.
_____________________________________________________
PRE_PARTITION  000      +00000000    Success.
_____________________________________________________

Summary of Partitioning Agents:
Rows Read                   = 100000
Rows Rejected               = 0
Rows Partitioned            = 100000
```

出力では、それぞれの出力データベース・パーティションで実行する load-to-file エージェントがあり、これらのエージェントが正常に実行されたことを示しています。コーディネーター・パーティションには事前パーティション化エージェントがあり、データベース・パーティション 1 で実行するパーティション化エージェントがあります。統計のサマリーでは、事前パーティション化エージェントによって 100,000 行が正常に読み取られ、パーティション化エージェントによって 100,000 行が正常に分散されたことを示しています。表には行がロードされなかったため、ロードされた行の数のサマリーは表示されません。

例 3

上記の PARTITION_ONLY ロード操作時に生成されたファイルをロードするには、以下のコマンドを発行します。

```
load from load.del of del replace into table1 partitioned db config mode
    load_only part_file_location /db/data
```

ロード・コマンドからの出力は、以下のようになります。

```
Agent Type     Node     SQL Code     Result
_____________________________________________________
LOAD           000      +00000000    Success.
_____________________________________________________
LOAD           001      +00000000    Success.
_____________________________________________________
LOAD           002      +00000000    Success.
_____________________________________________________
LOAD           003      +00000000    Success.
_____________________________________________________
RESULTS:       4 of 4 LOADs completed successfully.
_____________________________________________________

Summary of LOAD Agents:
Number of rows read         = 100000
Number of rows skipped      = 0
Number of rows loaded       = 100000
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 100000
```

出力では、それぞれの出力データベース・パーティションのロード・エージェントが正常に実行し、すべてのロード・エージェントによってロードされる行の数の合計が 100,000 であることを示しています。分散は実行されなかったため、分散される行のサマリーはありません。

例 4

以下の LOAD コマンドを発行したとします。

```
load from load.del of del replace into table1
```

ロード操作中に、ロード・データベース・パーティションの 1 つが表スペースでスペース不足になっているため、以下の出力が戻されます。

```
SQL0289N  Unable to allocate new pages in table space "DMS4KT".
SQLSTATE=57011

Agent Type     Node     SQL Code     Result
_____________________________________________________________________
LOAD           000      +00000000    Success.
_____________________________________________________________________
LOAD           001      -00000289    Error. May require RESTART.
_____________________________________________________________________
LOAD           002      +00000000    Success.
_____________________________________________________________________
LOAD           003      +00000000    Success.
_____________________________________________________________________
PARTITION      001      +00000000    Success.
_____________________________________________________________________
PRE_PARTITION  000      +00000000    Success.
_____________________________________________________________________
RESULTS:       3 of 4 LOADs completed successfully.
_____________________________________________________________________

Summary of Partitioning Agents:
Rows Read                   = 0
Rows Rejected               = 0
Rows Partitioned            = 0

Summary of LOAD Agents:
Number of rows read         = 0
Number of rows skipped      = 0
```

```
Number of rows loaded       = 0
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 0
```

出力では、ロード操作でエラー SQL0289 が戻されたことを示します。このデータベース・パーティションのサマリーでは、データベース・パーティション 1 でスペースが足りないことを示しています。データベース・パーティション 1 の表スペースのコンテナーにスペースが追加された場合、以下のようにしてロード操作を再始動できます。

```
load from load.del of del restart into table1
```

パーティション・データベース環境でのロード構成オプション:

パーティション・データベース環境でのロード操作を変更するために使用できる構成オプションが、多数存在します。

MODE *X*

複数パーティション・データベースをロードする際に実行するロード操作のモードを指定します。 PARTITION_AND_LOAD がデフォルトです。有効な値は以下のとおりです。

- PARTITION_AND_LOAD. データは (多くの場合は並列で) 分散され、それぞれ対応するデータベース・パーティションに同時にロードされます。
- PARTITION_ONLY. データは (多くの場合は並列で) 分散され、それぞれのロード・データベース・パーティションの指定したファイルに出力が書き込まれます。 CURSOR 以外のファイル・タイプの場合、各データベース・パーティションの出力ファイル名のフォーマットは *filename.xxx* です。ここで *filename* は、**LOAD** コマンドで指定された入力ファイル名で、*xxx* は 3 桁のデータベース・パーティション番号です。CURSOR ファイル・タイプの場合、各データベース・パーティションの出力ファイルの名前は PART_FILE_LOCATION オプションによって決められます。各データベース・パーティションの分散ファイルの位置を指定する方法の詳細については、 PART_FILE_LOCATION オプションを参照してください。

  注:

  1. このモードは CLI ロード操作には使用できません。
  2. 分散に必要な ID 列が表に収められている場合は、 **identityoverride** ファイル・タイプ修飾子が指定されない限り、このモードはサポートされません。
  3. ファイル・タイプ CURSOR 用に生成される分散ファイルは、DB2 の異なるリリース間で互換性がありません。これは、前のリリースで生成されたファイル・タイプ CURSOR の分散ファイルは、LOAD_ONLY モードを使用してロードできないということを意味します。同様に、現行リリースで生成されたファイル・タイプ CURSOR の分散ファイルは、将来のリリースでは LOAD_ONLY モードを使用してロードできません。
- LOAD_ONLY。 データはすでに分散されているものとします。この場合は分散プロセスが省略され、データはそれぞれ対応するデータベース・パーティションに同時にロードされます。 CURSOR 以外のファイル・タイプの場合、各データベース・パーティションの入力ファイル名のフォーマットは

filename.xxx となります。ここで *filename* は、**LOAD** コマンドで指定されたファイルの名前で、*xxx* は 3 桁のデータベース・パーティション番号です。CURSOR ファイル・タイプの場合、各データベース・パーティションの入力ファイルの名前は PART_FILE_LOCATION オプションによって決められます。各データベース・パーティションの分散ファイルの位置を指定する方法の詳細については、 PART_FILE_LOCATION オプションを参照してください。

注:

1. このモードは CLI ロード操作には使用できず、**LOAD** コマンドの **CLIENT** パラメーターが指定されている場合にも使用できません。
2. 分散に必要な ID 列が表に収められている場合は、 **identityoverride** ファイル・タイプ修飾子が指定されない限り、このモードはサポートされません。

- LOAD_ONLY_VERIFY_PART. データはすでに分散されているものとしますが、データ・ファイルにはパーティション・ヘッダーがありません。分散プロセスは省略され、データはそれぞれ対応するデータベース・パーティションに同時にロードされます。ロード操作時には、それぞれの行が正しいデータベース・パーティションにあることがチェックされます。 **dumpfile** ファイル・タイプ修飾子が指定されている場合には、データベース・パーティション違反のある行がダンプ・ファイルに入れられます。そうでなければ、行は廃棄されます。ロードしている特定のデータベース・パーティションにデータベース・パーティション違反がある場合、そのデータベース・パーティションのロード・メッセージ・ファイルに 1 つの警告が書き込まれます。各データベース・パーティションの入力ファイル名のフォーマットは *filename.xxx* となり、ここで *filename* は **LOAD** コマンドで指定されたファイルの名前で、*xxx* は 3 桁のデータベース・パーティション番号です。各データベース・パーティションの分散ファイルの位置を指定する方法の詳細については、PART_FILE_LOCATION オプションを参照してください。

  注:

  1. このモードは CLI ロード操作には使用できず、**LOAD** コマンドの **CLIENT** パラメーターが指定されている場合にも使用できません。
  2. 分散に必要な ID 列が表に収められている場合は、 **identityoverride** ファイル・タイプ修飾子が指定されない限り、このモードはサポートされません。

- ANALYZE。 すべてのデータベース・パーティションに均一に分散する最適な分散マップが生成されます。

PART_FILE_LOCATION *X*

PARTITION_ONLY、LOAD_ONLY、および LOAD_ONLY_VERIFY_PART モードでは、このパラメーターは、分散ファイルのロケーションを指定するために使用できます。このロケーションは、OUTPUT_DBPARTNUMS オプションによって指定される各データベース・パーティションに存在しなければなりません。指定されたロケーションが相対パス名の場合には、そのパスが現行ディレクトリーに追加されて、分散ファイルのロケーションが作成されます。

CURSOR ファイル・タイプの場合、このオプションを指定しなければならず、ロケーションは完全修飾されたファイル名を参照していなければなりません。この名前は、PARTITION_ONLY モードの場合は、各出力データベース・パーティショ

ンで作成された分散ファイルの完全修飾された基本ファイル名、または LOAD_ONLY モードの場合は、各データベース・パーティションから読み取ることのできるファイルのロケーションです。 PARTITION_ONLY モードの使用時に、ターゲット表中に LOB 列があると、指定した基本名のファイルが複数作成されることがあります。

CURSOR 以外のファイル・タイプの場合、このオプションが指定されないと、現行ディレクトリーが分散ファイルに使用されます。

OUTPUT_DBPARTNUMS *X*

X は、データベース・パーティション番号のリストを示します。データベース・パーティション番号は、ロード操作が実行されるデータベース・パーティションを示します。データベース・パーティション番号は、表が定義されているデータベース・パーティションのサブセットでなければなりません。デフォルトでは、すべてのデータベース・パーティションが選択されます。リストは括弧で囲まなければならず、リスト内のアイテムはコンマで区切らなければなりません。範囲を指定できます (例えば、(0, 2 to 10, 15))。

PARTITIONING_DBPARTNUMS *X*

X は、分散プロセスで使用されるデータベース・パーティション番号のリストを示します。リストは括弧で囲まなければならず、リスト内のアイテムはコンマで区切らなければなりません。範囲を指定できます (例えば、(0, 2 to 10, 15))。分散プロセスで指定されるデータベース・パーティションは、ロードされるデータベース・パーティションと異なっていてもかまいません。 PARTITIONING_DBPARTNUMS が指定されない場合には、最適なパフォーマンスを実現するために、ロード・ユーティリティーにより必要なデータベース・パーティションの数と使用するデータベース・パーティションが判別されます。

LOAD コマンドで **anyorder** ファイル・タイプ修飾子が指定されていない場合、ロード・セッションではパーティション化エージェントが 1 つだけ使用されます。さらに、OUTPUT_DBPARTNUMS オプションにデータベース・パーティションが 1 つだけ指定されている場合、またはロード操作のコーディネーター・パーティションが OUTPUT_DBPARTNUMS のエレメントではない場合、分散プロセスでロード操作のコーディネーター・パーティションが使用されます。その他の場合には、OUTPUT_DBPARTNUMS で最初のデータベース・パーティション (コーディネーター・パーティションではない) が分散プロセスで使用されます。

anyorder ファイル・タイプ修飾子が指定されている場合には、分散プロセスで使用されるデータベース・パーティションの数は、(OUTPUT_DBPARTNUMS のパーティションの数)/4 + 1 で決定されます。

MAX_NUM_PART_AGENTS *X*

ロード・セッションで使用されるパーティション化エージェントの最大数を指定します。デフォルトは 25 です。

ISOLATE_PART_ERRS *X*

個々のデータベース・パーティションで発生するエラーにロード操作がどのように対応するかを指示します。 **LOAD** コマンドで ALLOW READ ACCESS および **COPY YES** パラメーターの両方が指定される場合 (この場合のデフォルトは **NO_ISOLATION**) を除き、デフォルトは LOAD_ERRS_ONLY です。有効な値は以下のとおりです。

- SETUP_ERRS_ONLY。 セットアップ時にデータベース・パーティションにエラーが発生すると (データベース・パーティションへのアクセス時の障害、またはデータベース・パーティション上の表スペースまたは表へのアクセス時の障害など)、ロード操作は障害のあるデータベース・パーティションでは停止しますが、残りのデータベース・パーティションでは実行を継続します。データのロード中にデータベース・パーティションでエラーが発生すると、全体の操作が失敗します。
- LOAD_ERRS_ONLY。 セットアップ時にデータベース・パーティションにエラーが発生すると、ロード操作全体が失敗します。データのロード中にエラーが発生する場合、ロード操作はエラーが生じたデータベース・パーティションで停止します。残りのデータベース・パーティションでは、障害が発生するか、すべてのデータがロードされるまでロードが継続されます。新しくロードされたデータは、ロード再始動操作が実行されて正常に完了するまで表示されません。

  注: LOAD コマンドで **ALLOW READ ACCESS** および **COPY YES** パラメーターの両方が指定される場合には、このモードは使用できません。
- SETUP_AND_LOAD_ERRS。 このモードでは、セットアップまたはデータのロード時に生じるデータベース・パーティション・レベルのエラーによって、影響を受けたデータベース・パーティション上でのみ、処理が停止します。LOAD_ERRS_ONLY モードの場合と同様、データのロード中にパーティション・エラーが発生した場合、新しくロードされたデータはロード再始動操作が実行されて正常に完了するまで表示されません。

  注: LOAD コマンドで **ALLOW READ ACCESS** および **COPY YES** オプションの両方が指定される場合には、このモードは使用できません。
- NO_ISOLATION。 ロード操作時にエラーがあれば、ロード操作は失敗します。

STATUS_INTERVAL *X*
X は、読み取られたデータのボリュームを通知する頻度を示します。メジャー単位はメガバイト (MB) です。デフォルトは 100 MB です。有効な値は 1 から 4000 の整数です。

PORT_RANGE *X*
X は、内部通信のためのソケットの作成に使う TCP ポートの範囲を表します。デフォルトの範囲は 49152 から 65535 です。 **DB2ATLD_PORTS** レジストリー変数の値が起動時に定義される場合には、その値は PORT_RANGE ロード構成オプションの値で置き換えられます。 **DB2ATLD_PORTS** レジストリー変数の場合、範囲は以下のフォーマットで提供されます。

<lower-port-number:higher-port-number>

CLP からの場合は、以下のフォーマットです。

(lower-port-number, higher-port-number)

CHECK_TRUNCATION
プログラムが入出力時にデータ・レコードの切り捨てをチェックするように指定します。デフォルトの動作では、入出力時にはデータの切り捨てをチェックしません。

MAP_FILE_INPUT *X*
X は、分散マップの入力ファイル名を指定します。このパラメーターはカスタ

マイズされた分散マップの入ったファイルを示すため、分散マップがカスタマイズされている場合にはこのパラメーターを指定しなければなりません。カスタマイズされた分散マップを作成するには、**db2gpmap** プログラムを使用してデータベース・システム・カタログ表からマップを抽出するか、または、**LOAD** コマンドの ANALYZE モードを使用して最適なマップを生成します。ロード操作を継続するには、その前に ANALYZE モードを使用して生成されるマップをデータベース内のそれぞれのデータベース・パーティションに移動する必要があります。

MAP_FILE_OUTPUT *X*

X は、分散マップの出力ファイル名を示します。出力ファイルが作成されるのは、**LOAD** コマンドが発行されたデータベース・パーティションです。ただし、これは、そのデータベース・パーティションが、パーティション化の実行されるデータベース・パーティション・グループに関係している場合です。パーティション化に関係していないデータベース・パーティション (**PARTITIONING_DBPARTNUMS** によって定義される) 上で **LOAD** コマンドが呼び出された場合、出力ファイルは、**PARTITIONING_DBPARTNUMS** パラメーターを使って最初に定義されたデータベース・パーティションに作成されます。次のパーティション・データベース環境のセットアップを考慮してください。

```
1 serv1 0
2 serv1 1
3 serv2 0
4 serv2 1
5 serv3 0
```

serv3 で次の **LOAD** コマンドを実行すると、serv1 上に分散マップが作成されます。

```
LOAD FROM file OF ASC METHOD L ( ...) INSERT INTO table CONFIG
MODE ANALYZE PARTITIONING_DBPARTNUMS(1,2,3,4)
MAP_FILE_OUTPUT '/home/db2user/distribution.map'
```

このパラメーターは、ANALYZE モードが指定される際に使用しなければなりません。すべてのデータベース・パーティションに均一に分散する最適な分散マップが生成されます。このパラメーターが指定されておらず ANALYZE モードが指定されている場合には、プログラムは終了してエラーを戻します。

TRACE *X*

データ変換プロセスとハッシュ値の出力のダンプを調べることが必要になった場合にトレースするレコードの数を指定します。デフォルトは 0 です。

NEWLINE

入力データ・ファイルが、それぞれのレコードが改行文字によって区切られた ASC ファイルであり、**LOAD** コマンドで **reclen** ファイル・タイプ修飾子が指定されている場合に使用されます。このオプションが指定されると、それぞれのレコードの改行文字がチェックされます。また、**reclen** ファイル・タイプ修飾子で指定されたレコード長もチェックされます。

DISTFILE *X*

このオプションを指定すると、ロード・ユーティリティーは、指定された名前のデータベース・パーティション分散ファイルを生成します。データベース・パーティション分散ファイルには 32 768 個の整数が入っており、それぞれはターゲット表の分散マップの各項目に対応しています。ファイル内の各整数は、ロードされる入力ファイルの中で、対応する分散マップ項目にハッシュされる行数を表します。この情報はデータの中の歪度を識別するのに役立ちます。さらに、ユー

ティリティーの ANALYZE モードを使って表の新しい分散マップを生成すべきかどうか判断するのに役立ちます。このオプションを指定しない場合、ロード・ユーティリティーのデフォルト動作として、配布ファイルを生成しません。

注: このオプションを指定すると、最大で 1 つのパーティション化エージェントがロード操作のために使用されます。複数のパーティション化エージェントを明示的に要求した場合でも、1 つのみが使用されます。

OMIT_HEADER
分散ファイルに分散マップ・ヘッダーを組み込まないように指定します。指定されていなければ、ヘッダーが生成されます。

RUN_STAT_DBPARTNUM *X*
LOAD コマンドに **STATISTICS USE PROFILE** パラメーターが指定された場合には、1 つのデータベース・パーティションでのみ統計が収集されます。このパラメーターは、統計を収集するデータベース・パーティションを指定します。値が -1 か、またはまったく指定されない場合には、出力データベース・パーティション・リストの最初のデータベース・パーティションで統計が収集されます。

INGEST ユーティリティー

INGEST ユーティリティー (連続データ取り込み (INGEST)、または CDI とも呼ばれる) は、ファイルおよびパイプから DB2 のターゲット表にデータを流す、クライアント・サイドの高速な DB2 ユーティリティーです。INGEST ユーティリティーは、ターゲット表をロックせずに大量のリアルタイム・データを移動させることができるため、データの現行性と使用可能性とを両立させることができます。

INGEST ユーティリティーは、事前処理済みのデータを直接、または ETL ツールあるいはその他の手段により出力されたファイルから取り込みます (INGEST します)。継続的な実行が可能であるため、パイプを介する連続データ・ストリームを処理できます。データの取り込み (INGEST) 速度は、パーティション・データベース環境で大規模なデータベースへのデータ取り込みに使用できるほど高速です。

INGEST コマンドは、待ち時間が少なく、単一ステップでターゲット表を更新します。INGEST ユーティリティーは行ロッキングを使用するため、同じ表に対する他のユーザーの処理への干渉は、最小限に抑えられます。

このユーティリティーによって、ターゲット表をロッキングせずに、 SQL に類似のインターフェースを使用して表に対する DML 操作を実行できます。この INGEST 操作では、INSERT、UPDATE、MERGE、REPLACE、および DELETE の SQL ステートメントがサポートされます。INGEST ユーティリティーは、SQL 式を使用して、複数のデータ・フィールドから個別の列の値を作成する方法もサポートしています。

INGEST ユーティリティーの主な機能には、他に以下のものがあります。

- **時間または行数に基づくコミット。commit_count** INGEST 構成パラメーターを使用することにより、書き込まれた行数に基づいてコミットの頻度を決定したり、デフォルトの **commit_period** INGEST 構成パラメーターを使用することにより、指定した時間に基づいてコミットの頻度を決定したりすることができます。

- **拒否されたレコードのファイルまたは表へのコピー、または廃棄のサポート。** 拒否された行を **INGEST** コマンドがどのように処理するかを、INGEST ユーティリティー (**DUMPFILE** パラメーターを使用) または DB2 (**EXCEPTION TABLE** パラメーターを使用) で指定できます。
- **再開とリカバリーのサポート。**デフォルトで、すべての **INGEST** コマンドは最後のコミット・ポイントから再開可能です。さらに、**retry_count** INGEST 構成パラメーターが設定されている場合は、INGEST ユーティリティーによって、特定のエラーからのリカバリーが試行されます。

INGEST コマンドは、次の入力データ・フォーマットをサポートします。

- 区切りテキスト
- 定位置テキストおよびバイナリー
- 各種配列およびフォーマットの列

INGEST コマンドは、通常の表およびニックネームに加えて次の表タイプもサポートします。

- マルチディメンション・クラスタリング (MDC) 表および挿入時クラスタリング (ITC) 表
- 範囲パーティション表
- 範囲がクラスター化された表 (RCT)
- MAINTAINED BY USER として定義される、マテリアライズ照会表 (MQT)(サマリー表を含む)
- テンポラル表
- 更新可能なビュー (型付きビューを除く)

単一の **INGEST** コマンドは次の 3 つの主な段階を経て実行されます。

1. **転送**
   トランスポーターが、データ・ソースからデータを読み取り、レコードをフォーマッターのキューに渡します。INSERT および MERGE 操作の場合、入力ソースごとに 1 つのトランスポーター・スレッドが存在します (例: 入力ファイルごとに 1 つのスレッド)。UPDATE および DELETE 操作の場合、トランスポーター・スレッドは 1 つのみです。
2. **フォーマット**
   フォーマッターが、各レコードを解析し、データを DB2 データベース・システムに必要なフォーマットに変換し、フォーマット済みの各レコードを、そのレコードのパーティション用のフラッシャー・キューのいずれかに渡します。フォーマッター・スレッドの数は、 **num_formatters** 構成パラメーターで指定します。デフォルトは (論理 CPU 数)/2 です。
3. **フラッシュ**
   フラッシャーは、SQL ステートメントを実行して、DB2 表に対する操作を実行します。各パーティションのフラッシャーの数は、**num_flushers_per_partition** 構成パラメーターで指定します。デフォルトは max(1, ((number of logical CPUs)/2)/(number of partitions)) です。

取り込み (INGEST) 関連タスクの概要

このセクションでは、INGEST ユーティリティーの使用に関する主なセットアップ・タスクおよび運用タスクの全体像について説明します。

INGEST ミドルウェアをセットアップする

1. INGEST ユーティリティーの実行場所を決定する

   INGEST ジョブは、既存のマシン上で、またはスタンドアロン・マシンから実行できます。詳しくは、 137 ページの『INGEST ユーティリティーの実行場所を決定する』を参照してください。

2. INGEST ユーティリティー (DB2 Data Server Runtime Client および DB2 Data Server Client の一部) をインストールします。

   INGEST ユーティリティーを新規のスタンドアロン・マシンにインストールする場合は、DB2 クライアント・イメージのインストールを実行します。詳しくは、「*IBM データ・サーバー・クライアント機能 インストール*」の『IBM データ・サーバー・クライアントのインストール (Linux および UNIX)』を参照してください。

表にデータを取り込むプロセスを作成する

1. (必要な場合) コード・ページの問題を解決する

   入力データ、DB2 クライアント、および DB2 サーバーで同じコード・ページを使用しているかどうかによって、**INGEST** コマンドの実行前にユーザー処置が必要となる場合があります。詳しくは、 153 ページの『INGEST ユーティリティーのコード・ページに関する考慮事項』を参照してください。

2. 失敗した **INGEST** コマンドを再開できるようにセットアップする

   INGEST 操作を再開可能にするには、**INGEST** コマンドを発行する前に再開ログ表を作成する必要があります。詳しくは、 138 ページの『再開表の作成』を参照してください。

3. **INGEST** コマンドを入力する

   INGEST コマンドに、入力ソースやデータ・タイプなどの必須パラメーターと、さまざまなオプション・パラメーターを指定して実行します。コマンド構文および使用法の詳細および例については、「コマンド・リファレンス」の 139 ページの『データの INGEST』および 『INGEST』 を参照してください。

4. INGEST ジョブを絶え間なく継続的に処理するようにセットアップする

   事前に作成しておいた **INGEST** コマンドを簡単に呼び出せるようにするには、そのコマンドのスクリプトを作成し、必要な時に呼び出します。詳しくは、 158 ページの『シナリオ: INGEST ユーティリティーを使用して、継続的に送られてくるファイルを処理する』を参照してください。

運用タスクを実行する

- (必要な場合) 失敗した **INGEST** コマンドに対処する

INGEST ジョブが失敗した場合は、そのコマンドを再開するか終了するか選択します。詳しくは、 146 ページの『INGEST 操作が失敗した場合の再開方法』または 148 ページの『失敗した INGEST 操作の終了処理』を参照してください。

- **INGEST** コマンドをモニターする

  詳しくは、 149 ページの『INGEST 操作のモニター』を参照してください。

(オプション) パフォーマンスを最適化する

- **INGEST** コマンドの調整可能構成パラメーターを検討する
- ハイパフォーマンス要件を満たすために、**INGEST** コマンドを変更する

  詳しくは、 152 ページの『INGEST 操作のパフォーマンスに関する考慮事項』を参照してください。

INGEST ユーティリティーの実行場所を決定する

INGEST ユーティリティーは、DB2 クライアント・インストールの一部として含まれています。クライアントおよびサーバーのどちらからでも実行できます。

このタスクについて

INGEST ユーティリティーの実行場所としては、選択肢が 2 つあります。

データウェアハウス環境の既存サーバー上

このタイプのセットアップにおける INGEST ジョブの実行場所としては、選択肢が 2 つあります。

- DB2 コーディネーター・パーティション (アプリケーションの接続先であり、コーディネーター・エージェントがあるデータベース・パーティション・サーバー)
- 既存の ETL (抽出、変換、およびロード) サーバー

新規サーバー上

このタイプのセットアップにおける INGEST ジョブの実行場所としては、選択肢が 2 つあります。

- INGEST ユーティリティーのみを実行しているサーバー上
- INGEST ユーティリティー専用の追加の DB2 コーディネーター・パーティションもホスティングしているサーバー上

INGEST ユーティリティーのインストール場所の決定に影響を与える可能性がある要因がいくつかあります。

- パフォーマンス: INGEST ユーティリティーを専用のサーバーにインストールすることはパフォーマンスに大きな利益をもたらすため、大容量のデータ・セットを持つ環境には、これが最適と言えます。
- コスト: 既存のサーバーに INGEST ユーティリティーをインストールすれば、ユーティリティーの使用に伴う追加の費用は発生しません。
- 管理の容易さ

再開表の作成

失敗した **INGEST** コマンドは、デフォルトで、最後のコミット・ポイントから再開可能です。しかし、事前に再開表を作成しておく必要があります。この表には、**INGEST** コマンドの再開に必要な情報が保管されます。

このタスクについて

再開表の作成が必要になるのは 1 回だけです。データベース内のすべての **INGEST** コマンドはその表を使用します。

INGEST ユーティリティーは、この表を使用して、未完了の INGEST コマンドを最後のコミット・ポイントから再開するために必要な情報を保管します。

注: 再開表には、入力行のコピーが入っているわけではありません。どの行がコミット済みかを示すカウンターがいくつか入っているだけです。

制約事項

- 再開表は、INGEST ユーティリティーが更新するターゲット表と同じ表スペース内に配置することを推奨します。これができない場合は、再開表を含む表スペースが、ターゲット表を含む表スペースと必ず同じレベルになるようにしてください。例えば、どちらかの表スペースをリストアまたはロールフォワードする場合は、もう一方の表スペースも同じレベルにリストアまたはロールフォワードする必要があります。2 つの表スペースが異なるレベルにある場合、**INGEST** コマンドを **RESTART CONTINUE** オプション付きで実行すると、INGEST ユーティリティーが失敗するか、正しくないデータが INGEST されることがあります。
- 災害時リカバリー・ストラテジーに INGEST 操作のターゲット表の複製が含まれる場合、再開表とターゲット表の同期状態を維持するために、再開表の複製を行う必要もあります。

手順

再開表を作成するには、次のようにします。

- バージョン 10.1 サーバーを使用する場合は、SYSPROC.SYSINSTALLOBJECTS ストアード・プロシージャーを呼び出します。

```
db2 "CALL SYSPROC.SYSINSTALLOBJECTS('INGEST', 'C', tablespace-name, NULL)"
```

- バージョン 9.5、バージョン 9.7、または バージョン 9.8 サーバーを使用する場合は、次の SQL ステートメントを発行します。

```
CREATE TABLE SYSTOOLS.INGESTRESTART (
  JOBID             VARCHAR(256) NOT NULL,
  APPLICATIONID     VARCHAR(256) NOT NULL,
  FLUSHERID         INT          NOT NULL,
  FLUSHERDISTID     INT          NOT NULL,
  TRANSPORTERID     INT          NOT NULL,
  BUFFERID          BIGINT       NOT NULL,
  BYTEPOS           BIGINT       NOT NULL,
  ROWSPROCESSED     INT          NOT NULL,
    PRIMARY KEY (JOBID, FLUSHERID, TRANSPORTERID, FLUSHERDISTID))
  IN <tablespace-name>
  DISTRIBUTE BY (FLUSHERDISTID);

GRANT SELECT, INSERT, UPDATE, DELETE
   ON TABLE SYSTOOLS.INGESTRESTART TO PUBLIC;
```

タスクの結果

ここで、指定された表スペースに再開表 SYSTOOLS.INGESTRESTART を作成されるはずです。これにより、再開可能な **INGEST** コマンドを実行できるようになります。

例

DBA はすべての **INGEST** コマンドを再開可能なコマンドとして実行する計画なので、DBA は最初に再開表を作成しておく必要があります。

1. DBA は次のようにしてデータベースに接続します。
   ```
   db2 CONNECT TO sample
   ```
2. DBA は次のようにしてストアード・プロシージャーを呼び出します。
   ```
   db2 "CALL SYSPROC.SYSINSTALLOBJECTS('INGEST', 'C', NULL, NULL)"
   ```

次のタスク

再開表を変更するすべてのユーザーが適切な権限を持っていることを確認します。

- **INGEST** コマンドが RESTART NEW を指定する場合、ユーザーには再開表に対する SELECT、INSERT、UPDATE、および DELETE 特権が必要です。
- **INGEST** コマンドが RESTART TERMINATE を指定する場合、ユーザーには再開表に対する SELECT および DELETE 特権が必要です。

データの INGEST

INGEST ユーティリティーを使用すると、SQL による配列の挿入、更新、および削除を使用して、ソースのデータがなくなるまでデータを連続的に DB2 表に送り込むことができます。

始める前に

INGEST ユーティリティーを起動する前に、データをインポートするデータベースに接続する必要があります。

失敗した **INGEST** コマンドは、デフォルトで、最後のコミット・ポイントから再開可能です。しかし、あらかじめ再開表を作成しておく必要があります。そうでないと、実行したコマンドを再開できないというエラー・メッセージが出されます。INGEST ユーティリティーは、この表を使用して、未完了の **INGEST** コマンドを最後のコミット・ポイントから再開するために必要な情報を保管します。これについて詳しくは、138 ページの『再開表の作成』を参照してください。

このタスクについて

必要な特権および権限のリストを確認する場合には、**INGEST** コマンド権限を参照してください。

制約事項

INGEST ユーティリティーにおける制限の包括的なリストを確認する場合には、150 ページの『INGEST ユーティリティーの制約事項および制限事項』を参照してください。

手順

以下のように、少なくとも、ソース、フォーマット、およびターゲット表を指定して、**INGEST** コマンドを実行します。

```
INGEST FROM FILE my_file.txt
    FORMAT DELIMITED
    INSERT INTO my_table;
```

INGEST コマンドには、以下のように、**RESTART NEW** パラメーターを含むストリングも指定することをお勧めします。

```
INGEST FROM FILE my_file.txt
   FORMAT DELIMITED
   RESTART NEW 'CDIjob001'
   INSERT INTO my_table;
```

指定するストリングの長さは最高 128 バイトです。このストリングは、 **INGEST** コマンドを一意的に識別するものであるため、 **RESTART NEW** オプションが指定されていて、まだ完了していない、現行データベース内の全 **INGEST** コマンドにおいて固有でなければなりません。

例

基本的な INGEST の例

次の例では、区切り文字で区切られているテキスト・ファイルからデータを挿入します。

```
INGEST FROM FILE my_file.txt
    FORMAT DELIMITED
    INSERT INTO my_table;
```

次の例では、フィールドがコンマ (デフォルト) で区切られているテキスト・ファイルからデータを挿入します。ファイル内の各フィールドは、表の各列に対応しています。

```
INGEST FROM FILE my_file.txt
   FORMAT DELIMITED
   (
      $field1 INTEGER EXTERNAL,
      $field2 DATE 'mm/dd/yyyy',
      $field3 CHAR(32)
   )
   INSERT INTO my_table
      VALUES($field1, $field2, $field3);
```

区切り文字のオーバーライドの例

次の例では、前の例と類似したデータを挿入していますが、垂直バーを使用してフィールドが区切られています。

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED by '|'
 (
  $field1 INTEGER EXTERNAL,
  $field2 DATE 'mm/dd/yyyy',
  $field3 CHAR(32)
 )
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

フィールド定義および VALUES リストを省略する例

次の例では、表は以下のように定義されています。

```
CREATE TABLE my_table (
    c1 VARCHAR(32),
    c2 INTEGER GENERATED BY DEFAULT AS IDENTITY,
    c3 INTEGER GENERATED ALWAYS AS (c2 + 1),
   );
```

ユーザーは次の **INGEST** コマンドを発行します。

```
INGEST FROM FILE my_file.txt
    FORMAT DELIMITED
    INSERT INTO mytable;
```

- デフォルトのフィールド定義リストは次のようになります。

  ```
  (
      $C1 CHARACTER(32),
      $C2 INTEGER EXTERNAL,
      $C3 INTEGER EXTERNAL
  )
  ```

- INSERT ステートメントのデフォルトの VALUES リストは、次のとおりです。

  ```
  VALUES($C1, $C2, DEFAULT)
  ```

  フィールド $C3 に対応する列は GENERATED ALWAYS と定義されているため、 3 番目の値が DEFAULT になっていることに注意してください。4 番目の値は、フィールドがないため省略されています。

余分なフィールドを使用して列値を計算する例

次の例は、区切り文字のオーバーライドの例と同じですが、最初の 2 つのフィールドのみが、表の最初の 2 列 (PROD_ID および DESCRIPTION) に対応していて、表の 3 番目の列 (TOTAL_PRICE) の値は、残りの 3 つのフィールドから計算しています。

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED BY '|'
 (
  $prod_ID     CHAR(8),
  $description CHAR(32),
  $price       DECIMAL(5,2) EXTERNAL,
  $sales_tax   DECIMAL(4,2) EXTERNAL,
  $shipping    DECIMAL(3,2) EXTERNAL
 )
 INSERT INTO my_table(prod_ID, description, total_price)
  VALUES($prod_id, $description, $price + $sales_tax + $shipping);
```

充てん文字フィールドの例

次の例では、フィールドがコンマ (デフォルト) で区切られているテキスト・ファイルからデータを挿入します。ファイル内の各フィールドは、列 2 と 3 の間、および列 3 と 4 の間に余分なフィールドが存在している点を除けば、表の各列に対応しています。

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $field1  INTEGER,
  $field2  CHAR(8),
  $filler1 CHAR,
  $field3  CHAR(32),
  $filler2 CHAR,
  $field4  DATE
 )
 INSERT INTO my_table VALUES($field1, $field2, $field3, $field4);
```

形式修飾子の例

次の例では、区切り文字で区切られている、コード・ページ 850 のテキスト・ファイルからデータを挿入しています。日付フィールドは米国形式であり、文字フィールドは等号で囲まれています。

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 INPUT CODEPAGE 850
 (
  $field1 INTEGER EXTERNAL,
  $field2 DATE 'mm/dd/yyyy',
  $field3 CHAR(32) ENCLOSED BY '='
 )
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

定位置の例

次の例では、指定された位置に各フィールドが存在するファイルから、データを挿入しています。ファイル内の各フィールドは、表の各列に対応しています。

```
INGEST FROM FILE my_file.txt
    FORMAT POSITIONAL
 (
  $field1 POSITION(1:8)   INTEGER EXTERNAL,
  $field2 POSITION(10:19) DATE 'yyyy-mm-dd',
  $field3 POSITION(25:34) CHAR(10)
 )
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

DEFAULTIF の例

この例は、前の例に類似していますが、 2 番目のフィールドの先頭がブランクである場合に、 INGEST ユーティリティーがデフォルト値を挿入する点が異なっています。

```
INGEST FROM FILE my_file.txt
 FORMAT POSITIONAL
 (
  $field1 POSITION(1:8) INTEGER EXTERNAL,
  $field2 POSITION(10:19) DATE 'yyyy-mm-dd' DEFAULTIF = ' ',
    $field3 POSITION(25:34) CHAR(10)
 )
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

この例は、前の例と同じですが、データ列の後の列に、デフォルト・インディケーターがある点が異なっています。

```
INGEST FROM FILE my_file.txt
 FORMAT POSITIONAL
 (
  $field1 POSITION(1:8)   INTEGER EXTERNAL,
  $field2 POSITION(10:19) DATE 'yyyy-mm-dd' DEFAULTIF(35) = ' ',
    $field3 POSITION(25:34) CHAR(10)
 )
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

複数入力ソースの例

次の例では、区切り文字で区切られている 3 つのテキスト・ファイルからデータを挿入します。

```
INGEST FROM FILE my_file.txt, my_file2.txt, my_file3.txt
 FORMAT DELIMITED
 (
  $field1 INTEGER EXTERNAL,
  $field2 DATE 'mm/dd/yyyy',
  $field3 CHAR(32)
 )
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

パイプの例

この例では、パイプからデータを挿入します。

```
INGEST FROM PIPE my_pipe
 FORMAT DELIMITED
 (
  $field1 INTEGER EXTERNAL,
  $field2 DATE 'mm/dd/yyyy',
  $field3 CHAR(32)
 )
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

オプションの例

この例では、フィールドがコンマ (デフォルト) で区切られているテキスト・ファイルからデータを挿入します。ファイル内の各フィールドは、表の各列に対応しています。このコマンドは、(制約違反などによって) DB2 から拒否された行を表 EXCP_TABLE に書き込み、他のエラーによって拒否された行を破棄し、メッセージをファイル messages.txt に書き込むように指定しています。

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $field1 INTEGER EXTERNAL,
  $field2 DATE 'mm/dd/yyyy',
  $field3 CHAR(32)
 )
 EXCEPTION TABLE excp_table
    MESSAGES messages.txt
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

再開の例

この例では、INGEST ジョブ ID を指定して、**INGEST** コマンド (デフォルトで再開可能) を実行しています。

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $field1 INTEGER EXTERNAL,
  $field2 DATE 'mm/dd/yyyy',
  $field3 CHAR(32)
 )
 RESTART NEW 'ingestcommand001'
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

このコマンドが完了せずに終了した場合は、次のコマンドを使用して再開することができます。

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $field1 INTEGER EXTERNAL,
```

```
 $field2 DATE 'mm/dd/yyyy',
 $field3 CHAR(32)
)
RESTART CONTINUE 'ingestcommand001'
INSERT INTO my_table
 VALUES($field1, $field2, $field3);
```

再開を終了させる例

この例では、前述の『再開の例』と同じ **INGEST** コマンドを実行しています。

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $field1 INTEGER EXTERNAL,
  $field2 DATE 'mm/dd/yyyy',
  $field3 CHAR(32)
 )
 RESTART NEW 'ingestcommand001'
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

このコマンドが完了せずに終了した場合に、再開させる予定がなければ、次のコマンドを使用して、再開レコードをクリーンアップできます。

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $field1 INTEGER EXTERNAL,
  $field2 DATE 'mm/dd/yyyy',
  $field3 CHAR(32)
 )
 RESTART TERMINATE 'ingestcommand001'
 INSERT INTO my_table
  VALUES($field1, $field2, $field3);
```

このコマンドを実行すると、ジョブ ID「ingestcommand001」を指定して **INGEST** コマンドを再開できなくなりますが、このストリングは、新しい **INGEST** コマンドの RESTART NEW パラメーターに再利用できるようになります。

列の再配列の例

この例では、フィールドがコンマで区切られているテキスト・ファイルからデータを挿入します。表には 3 つの列があり、入力データ内の各フィールドは、表の列とは逆の順序で並んでいます。

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $field1 INTEGER EXTERNAL,
  $field2 DATE 'mm/dd/yyyy',
  $field3 CHAR(32)
 )
 INSERT INTO my_table
  VALUES($field3, $field2, $field1);
```

基本的な UPDATE、MERGE、および DELETE の例

次の例では、主キーが、入力ファイル内の対応するフィールドと一致する場合に、その表の行を更新します。

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $key1  INTEGER EXTERNAL,
```

```
 $key2  INTEGER EXTERNAL,
 $data1 CHAR(8),
 $data2 CHAR(32),
 $data3 DECIMAL(5,2) EXTERNAL
 )
UPDATE my_table
 SET (data1, data2, data3) = ($data1, $data2, $data3)
 WHERE (key1 = $key1) AND (key2 = $key2);
```

または

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $key1  INTEGER EXTERNAL,
  $key2  INTEGER EXTERNAL,
  $data1 CHAR(8),
  $data2 CHAR(32),
  $data3 DECIMAL(5,2) EXTERNAL
  )
 UPDATE my_table
  SET data1 = $data1, data2 = $data2, data3 = $data3
  WHERE (key1 = $key1) AND (key2 = $key2);
```

この例では、入力ファイル内のデータを、ターゲット表にマージします。主キー・フィールドが表の行と一致する入力行については、その入力行を使用して表の行を更新します。他の入力行については、表に追加します。

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $key1  INTEGER EXTERNAL,
  $key2  INTEGER EXTERNAL,
  $data1 CHAR(8),
  $data2 CHAR(32),
  $data3 DECIMAL(5,2) EXTERNAL
  )
 MERGE INTO my_table
  ON (key1 = $key1) AND (key2 = $key2)
  WHEN MATCHED THEN
   UPDATE SET (data1, data2, data3) = ($data1, $data2, $data3)
  WHEN NOT MATCHED THEN
   INSERT VALUES($key1, $key2, $data1, $data2, $data3);
```

この例では、主キーが、入力ファイル内の対応する主キー・フィールドと一致する場合に、その表の行を削除します。

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $key1  INTEGER EXTERNAL,
  $key2  INTEGER EXTERNAL
  )
 DELETE FROM my_table
  WHERE (key1 = $key1) AND (key2 = $key2);
```

複合 SQL の例

列 KEY、DATA、および ACTION を持つ表が存在する次の例を考えてみましょう。次のコマンドは、主キー列 (KEY) が、入力ファイル内の対応するフィールドと一致していて、 ACTION 列が「U」である場合に、その表の行の DATA 列を更新します。

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
```

```
 $key_fld  INTEGER EXTERNAL,
 $data_fld INTEGER EXTERNAL
)
UPDATE my_table
 SET data = $data_fld
  WHERE (key = $key_fld) AND (action = 'U');
```

次の例は、前の例と同じですが、キーが一致していて ACTION 列が「D」である場合に、その行を表から削除する点が異なっています。

```
INGEST FROM FILE my_file.txt
 FORMAT DELIMITED
 (
  $key_fld  INTEGER EXTERNAL,
  $data_fld INTEGER EXTERNAL
 )
 MERGE INTO my_table
  ON (keyl = $key_fld)
  WHEN MATCHED AND (action = 'U') THEN
   UPDATE SET data = $data_fld
  WHEN MATCHED AND (action = 'D') THEN
   DELETE;
```

次のタスク

INGEST コマンドが正常に完了した場合、その **RESTART NEW** パラメーターに指定したストリングは再利用可能です。

INGEST コマンドが失敗して、再開する場合は、元のコマンドで指定したストリングに **RESTART CONTINUE** オプションを付けて指定する必要があります。

失敗した **INGEST** コマンドを再開せず、再開表の項目をクリーンアップする場合は、**RESTART TERMINATE** オプションを指定して **INGEST** コマンドを再実行します。

INGEST 操作が失敗した場合の再開方法:

INGEST コマンドが完了せずに失敗した場合に、これを再開させるには、**INGEST** コマンドに RESTART CONTINUE オプションを指定して再実行します。この 2 回目の **INGEST** コマンドは、最後のコミット・ポイントから処理を開始します。このコマンドも再開可能です。

始める前に

失敗した **INGEST** コマンドを再開させるユーザー ID は、再開ログ表に対する SELECT、INSERT、UPDATE、および DELETE 特権を持っていなければなりません。

このタスクについて

INGEST ユーティリティーは、ファイルまたはパイプの最後に到達すると、コマンドが完了したものと判断します。これ以外の状況では、**INGEST** ユーティリティーは、コマンドは完了しなかったと見なします。例えば、次のような場合があります。

- 入力ファイルまたはパイプの読み取り中に、 **INGEST** コマンドが入出力エラーを受け取る。
- **INGEST** コマンドが、DB2 データベース・システムからクリティカルなシステム・エラーを受け取る。

- **INGEST** コマンドが、 **INGEST** コマンド内の SQL ステートメントの処理を妨害するような DB2 データベース・システム・エラー (対象の表がもう存在していないなど) を受け取る。
- **INGEST** コマンドが強制終了されるか、または異常終了する。

制約事項

- ターゲット表と再開表が異なる表スペースにある場合、それら 2 つの表スペースは、ロールフォワード操作であるかリストア操作であるかという観点において同じレベルのものである必要があります。
- ターゲット表との同期を維持するために表全体をリストアする場合を除き、再開表の内容を変更することはできません。
- **num_flushers_per_partition** 構成パラメーターは、元のコマンドのものと同一でなければなりません。
- 入力がファイルまたはパイプからの場合、入力ファイル数または入力パイプ数が、元のコマンドのものと同一でなければなりません。
- 入力のファイル、またはパイプは、元のコマンドの場合と同じレコードを同じ順序で提供しなければなりません。
- 以下の **INGEST** コマンド・パラメーターは、元のコマンドのパラメーターと同一でなければなりません。
  - 入力タイプ (ファイルまたはパイプ)
  - SQL ステートメント
  - フィールド定義リスト (フィールドの数、およびすべてのフィールド属性を含む)
- SQL コマンドが更新するターゲット表の列には、元のコマンドの実行時と同じ定義がなければなりません。
- パーティション・データベース環境内で、データベース・パーティションが追加または削除されていてはなりません。
- パーティション・データベース環境内のパーティション間でデータが再配分されていてはなりません。
- **INGEST** コマンドで DUMPFILE (BADFILE) パラメーターを指定すると、**INGEST** コマンドが 1 回の実行で完了した場合のみ、ダンプ・ファイルが完結していることが保証されます。**INGEST** コマンドが失敗し、再開させたコマンドが成功した場合、これら 2 回のコマンドで生成されたダンプ・ファイルの両方を合わせても、いくつかのレコードが欠落していたり、重複レコードが含まれていたりする可能性があります。

3 番目、4 番目、5 番目、または 9 番目の制約事項に違反すると、INGEST ユーティリティーはエラーを出し、**INGEST** コマンドを終了します。その他の制約事項の場合、INGEST ユーティリティーはエラーを出しませんが、再開された **INGEST** コマンドは、元のコマンドが完了していた場合のものとは異なる出力行を生成する可能性があります.

手順

失敗した **INGEST** 操作を再開するには、以下を行います。

1. 有効な情報を使用して、失敗の原因となった問題を診断し、その問題を解決します。
2. **RESTART CONTINUE** オプションを、該当するジョブ ID と一緒に指定して、**INGEST** コマンドを再実行します。

タスクの結果

再開した **INGEST** コマンドが完了した場合、そのジョブ ID は、今後の **INGEST** コマンドに再利用できます。

例

次の **INGEST** コマンドが失敗しました。

```
INGEST FROM FILE my_file.txt
   FORMAT DELIMITED
   (
      $field1 INTEGER EXTERNAL,
      $field2 DATE 'mm/dd/yyyy',
      $field3 CHAR(32)
   )
   RESTART NEW 'ingestjob001'
   INSERT INTO my_table
      VALUES($field1, $field2, $field3);
```

DBA は、失敗の原因となった問題を解決し、次のコマンドを使用して、**INGEST** コマンドを再開します (この場合は、最後のコミット・ポイントから処理が開始されます)。

```
INGEST FROM FILE my_file.txt
   FORMAT DELIMITED
   (
      $field1 INTEGER EXTERNAL,
      $field2 DATE 'mm/dd/yyyy',
      $field3 CHAR(32)
   )
   RESTART CONTINUE 'ingestjob001'
   INSERT INTO my_table
      VALUES($field1, $field2, $field3);
```

失敗した INGEST 操作の終了処理:

INGEST コマンドが完了せずに失敗した場合に、再開させる必要がなければ、その **INGEST** コマンドに **RESTART TERMINATE** オプションを指定して再実行します。このコマンド・オプションは、失敗した **INGEST** コマンドに関するログ・レコードをクリーンアップします。

始める前に

失敗した **INGEST** コマンドを終了させるユーザー ID は、再開ログ表に対する SELECT および DELETE 特権を持っていなければなりません。

手順

失敗した **INGEST** 操作を終了するには、**INGEST** コマンドを再発行します。 **RESTART TERMINATE** パラメーターに適切なストリングを付けて指定します。

タスクの結果

再開された **INGEST** コマンドが完了すると、その **RESTART NEW** ストリングを次回の **INGEST** コマンドに再利用できます。

例

次の **INGEST** コマンドが失敗しました。

```
INGEST FROM FILE my_file.txt
   FORMAT DELIMITED
   (
      $field1 INTEGER EXTERNAL,
      $field2 DATE 'mm/dd/yyyy',
      $field3 CHAR(32)
   )
   RESTART NEW 'ingestjob001'
   INSERT INTO my_table
      VALUES($field1, $field2, $field3);
```

DBA は **INGEST** コマンドを再開しないため、次のコマンド (**RESTART TERMINATE** パラメーターを含む) を使用してこのコマンドを終了します。

```
INGEST FROM FILE my_file.txt
   FORMAT DELIMITED
   (
      $field1 INTEGER EXTERNAL,
      $field2 DATE 'mm/dd/yyyy',
      $field3 CHAR(32)
   )
   RESTART TERMINATE 'ingestjob001'
   INSERT INTO my_table
      VALUES($field1, $field2, $field3);
```

INGEST 操作のモニター

INGEST LIST コマンドまたは **INGEST GET STATS** コマンドを使用して、**INGEST** コマンドの進行状況をモニターすることができます。

始める前に

INGEST LIST コマンドと **INGEST GET STATS** コマンドを実行するためには、別個の CLP セッションが必要ですが、それらのセッションは、**INGEST** コマンドを実行しているのと同じマシン上で実行する必要があります。

手順

INGEST 操作をモニターする方法がいくつかあります。

- 現在実行中のすべての **INGEST** コマンドに関する基本情報を得るには、**INGEST LIST** コマンドを使用します。
- 特定の **INGEST** コマンド、または現在実行中のすべての **INGEST** コマンドに関する、より詳細な情報を得るには、**INGEST GET STATS** コマンドを使用します。
- MON_GET_CONNECTION 表関数などのインターフェースを使用して、以下のモニター・エレメントを照会することもできます。
  - **client_acctng**
  - **client_applname**
  - **appl_name**

- **client_userid**
- **client_wrkstnname**

例

以下に、**INGEST LIST** コマンドによる出力の例を示します。

```
INGEST LIST

Ingest job ID       = DB21000:20101116.123456.234567:34567:45678
Ingest temp job ID  = 1
Database Name       = MYDB
Input type          = FILE
Target table        = MY_SCHEMA.MY_TABLE
Start Time          = 04/10/2010 11:54:45.773215
Running Time        = 01:02:03
Number of records processed = 30,000
```

以下に、**INGEST GET STATS** コマンドによる出力の例を示します。

```
INGEST GET STATS FOR 4

Ingest job ID = DB21000:20101116.123456.234567:34567:4567
Database      = MYDB
Target table  = MY_SCHEMA.MY_TABLE1

Records/sec      Flushes/sec       Records/sec     Flushes/sec
since start      since start       since last      since last       Total records
---------------- ----------------- --------------- ---------------- -------------
54321            65432             76543           87654            98765
```

以下は、MON_GET_CONNECTION 表関数を使用して、変更された行の数、およびコミットの数を得る方法の例です。

```
SELECT client_acctng AS "Job ID",
      SUM(rows_modified) AS "Total rows modified",
      SUM(total_app_commits) AS "Total commits"
   FROM TABLE(MON_GET_CONNECTION(NULL, NULL))
   WHERE application_name = 'DB2_INGEST'
   GROUP BY client_acctng
   ORDER BY 1

Job ID                                          Total rows modified Total commits
----------------------------------------------- ------------------- -------------
DB21000:20101116.123456.234567:34567:45678                       92            52
DB21000:20101116.987654.234567:34567:45678                      172           132

  2 record(s) selected.
```

INGEST ユーティリティーの制約事項および制限事項

INGEST ユーティリティーの使用については、注意すべき制約事項がいくつかあります。

再開可能性

- 入力データ・ソースのタイプが変更されると、INGEST ユーティリティーがその変更を検出できず、元の失敗したコマンドとは異なる出力行を生成することがあります。

表のサポート

- INGEST ユーティリティーは、DB2 for Linux, UNIX and Windows の表に対する操作のみをサポートします。

- INGEST ユーティリティーは、次に対する操作はサポートしません。
  - 作成済みまたは宣言済みのグローバル一時表
  - 型付き表
  - 型付きビュー

入力タイプ、入力フォーマット、および列のタイプ

- INGEST ユーティリティーは、次の列タイプをサポートしません。
  - ラージ・オブジェクト・タイプ (LOB、BLOB、CLOB、DBCLOB)
  - XML
  - 構造化タイプ
  - 上記でリストされたいずれかのタイプに基づくユーザー定義のデータ型を持つ列
- さらに、INGEST ユーティリティーには、生成された列に関する以下の制限があります。
  - INGEST ユーティリティーは、GENERATED ALWAYS と定義されている列に、値を割り当てることはできません。**INGEST** コマンドでの SQL ステートメントが INSERT または UPDATE で、ターゲット表に GENERATED ALWAYS 列がある場合、以下のいずれかを実行しない限り、この挿入操作または更新操作は失敗し (SQL0798N)、**INGEST** コマンドは終了します。
    - アップデートする列のリストから、その列を削除する。
    - INSERT または UPDATE ステートメントで、その列に割り当てる値として DEFAULT を指定する。
  - INGEST ユーティリティーは、デフォルト値と特定の値の組み合わせを、GENERATED BY DEFAULT AS IDENTITY として定義されている列に割り当てることはできません。**INGEST** コマンドに対する SQL ス**INGEST** コマンドでの SQL ステートメントが INSERT または UPDATE で、ターゲット表に GENERATED BY DEFAULT AS IDENTITY 列がある場合、以下のいずれかを実行しない限り、この挿入操作または更新操作は失敗し (SQL0407N)、**INGEST** コマンドはレコードを拒否します。
    - アップデートする列のリストから、その列を削除する。
    - INSERT または UPDATE ステートメントで、その列に割り当てる値として DEFAULT を指定する。
    - 列に割り当てる値として、決して NULL に評価されない式を指定する。例えば、式を $field1 とすると、$field1 は入力レコード内で決して NULL 値をとりません。

他の DB2 フィーチャーをINGEST ユーティリティーと共に使用する場合に関連した制限

- **CONNECT_MEMBER** パラメーターを別にすると、**SET CLIENT** コマンド (接続設定用) は、INGEST ユーティリティーの接続動作に影響を与えません。
- **LIST HISTORY** コマンドでは、 INGEST 操作に関する表示は出力されません。

- **SET UTIL_IMPACT_PRIORITY** コマンドは、 **INGEST** コマンドに影響を与えません。
- **util_impact_lim** データベース・マネージャー構成パラメーターは、**INGEST** コマンドに影響を与えません。
- CURRENT SCHEMA、CURRENT TEMPORAL SYSTEM_TIME、および CURRENT TEMPORAL BUSINESS_TIME は別にして、INGEST ユーティリティーは、SQL ステートメントの実行に影響するほとんどの特殊レジスターの設定を無視します。

一般的な INGEST ユーティリティーの制約

- 複数の基本表を持つビューに INGEST する場合、セキュリティー・ポリシーによって保護される基本表は、同一のセキュリティー・ポリシーによって保護する必要があります。(一部の基本表を保護されないままにしておくことはできますが、保護される基本表には同一のセキュリティー・ポリシーを使用する必要があります。)

ニックネームのサポート

- **INGEST** コマンドで RESTART NEW または RESTART CONTINUE オプションが設定されているか、デフォルトでこれらのオプションになる場合で、なおかつターゲット表がニックネームであるか、またはニックネームを更新する更新可能ビューである場合は、ニックネームを含むサーバー定義の DB2_TWO_PHASE_COMMIT サーバー・オプションが 'Y' に設定されていることを確認してください。
- **INGEST** コマンドを発行する前に、**SET SERVER OPTION** を使用して 2 フェーズ・コミットを使用可能にすることはできません。これは、そのコマンドが CLP 接続に対してのみ有効であるのに対し、**INGEST** コマンドは独自の接続を確立するためです。カタログ内のサーバー定義で、サーバー・オプションを設定する必要があります。
- データベース・パーティション・フィーチャーでは DB2_TWO_PHASE_COMMIT サーバー・オプションを使用できません。つまり、パーティション・データベース環境モードと、再開可能 INGEST コマンド、およびニックネームへの INGEST の組み合わせはサポートされません。
- ニックネームに対して実行した場合、このユーティリティーのパフォーマンス上のメリットが減少します。

INGEST 操作に関する追加の考慮事項

INGEST 操作のパフォーマンスに関する考慮事項

以下の一連のガイドラインを使用して、INGEST ジョブのパフォーマンスを調整します。

フィールド・タイプおよび列タイプ

関連する列タイプと同じタイプにフィールドを定義します。両者のタイプが異なっていると、INGEST ユーティリティーまたは DB2 が入力データを列タイプに変換しなければなりません。

マテリアライズ照会表 (MQT)

REFRESH IMMEDIATE と定義されている MQT の基本表である表にデータを INGEST する場合、MQT を更新するための時間が必要なため、パフォーマンスが大幅に低下する可能性があります。

行サイズ

行サイズの小さな表では、**commit_count** INGEST 構成パラメーターの設定を増やします。行サイズの大きな表では、**commit_count** INGEST 構成パラメーターの設定を減らします。

その他のワークロード

INGEST ユーティリティーを他のワークロードと一緒に実行している場合は、**locklist** データベース構成パラメーターの設定を増やし、**commit_count** INGEST 構成パラメーターの設定を減らします。

INGEST ユーティリティーのコード・ページに関する考慮事項

INGEST ユーティリティーで入力データを処理する際には、アプリケーション (クライアント) のコード・ページ、入力データのコード・ページ、データベースのコード・ページという 3 つのコード・ページが関係します。

コード・ページ	指定方法	デフォルト
アプリケーション (クライアント) コード・ページ (CLP コマンド・ファイルで使用される)	現行ロケールから決まる	現行ロケールから決まる
入力データのコード・ページ	**INGEST** コマンドの INPUT CODEPAGE	アプリケーションのコード・ページ
データベース・コード・ページ	**CREATE DATABASE** コマンドで指定される	1208 (Unicode の UTF-8 エンコード)

入力データのコード・ページがアプリケーションのコード・ページと異なる場合、INGEST ユーティリティーは、アプリケーションのコード・ページを一時的にオーバーライドして、入力データのコード・ページに変更します。これによって、DB2 は、入力データのコード・ページからデータベースのコード・ページに直接データを変換できるようになります。いくつかの状況では、INGEST ユーティリティーは、アプリケーション・コード・ページをオーバーライドできません。このような場合、INGEST ユーティリティーは、 FOR BIT DATA として定義されていない文字データをアプリケーション・コード・ページに変換してから、 DB2 に引き渡します。どのような場合であっても、列が FOR BIT DATA として定義されていなければ、DB2 はそのデータをデータベース・コード・ページに変換します。

CLP コマンド・ファイルのコード・ページ

16 進数の定数を除き、INGEST ユーティリティーは、**INGEST** コマンドのテキストを、アプリケーション・コード・ページのものと想定します。

INGEST ユーティリティーは、**INGEST** コマンドに指定されたストリングを比較する必要がある場合 (DEFAULTIF 文字と入力データ内の文字との比較など)、それらの比較対象ストリングが同じコード・ページのものとなるように、必要なコード・ページ変換を実行します。 INGEST ユーティリティーまたは DB2 のいずれも、 16 進定数の変換は行いません。

入力データのコード・ページ

フィールドと、フィールドの割り当て先の表の列の両方が、FOR BIT DATA として定義されている場合、INGEST ユーティリティーまたは DB2 のいずれも、コード・ページ変換を行いません。例えば、**INGEST** コマンドでフィールド $c1 を列 C1 に割り当て、この両方を CHAR FOR BIT DATA として定義するとします。入力フィールドに X'E9' が入っている場合、DB2 は、入力データ・コード・ページまたはデータベース・コード・ページとは無関係に、列 C1 に X'E9' を設定します。

列の定義に FOR BIT DATA が省略されている場合は、対応するフィールド定義でも FOR BIT DATA を省略することを強くお勧めします。同様に、列の定義に FOR BIT DATA が指定されている場合は、対応するフィールドでも FOR BIT DATA を指定する必要があります。そうしないと、INGEST ユーティリティーがアプリケーション・コード・ページをオーバーライドできるかどうかによって、列に割り当てられる値が異なるため、予測不能になります。

以下の例は、この状態を示しています。

- 入力データ・コード・ページが 819。
- アプリケーション・コード・ページは 850 です。
- データベース・コード・ページは 1208 (UTF-8)
- 入力データは「é」(揚音アクセント付きの「e」) であり、これは、コード・ページ 819 では X'E9'、コード・ページ 850 では X'82'、UTF-8 では X'C3A9' です。

次の表は、フィールドまたは列、またはその両方が FOR BIT DATA と定義されているかどうか、および INGEST ユーティリティーがアプリケーション・コード・ページをオーバーライドできるかどうかによって、サーバー上の最終的なデータがどのようになるかを示しています。

表 16. フィールドおよび列の定義に FOR BIT DATA と指定されている場合の考えられる結果

フィールド定義	列定義	入力データ (コード・ページ 819)	INGEST ユーティリティーによりアプリケーションのコード・ページ 850 に変換された後のデータ	INGEST ユーティリティーがアプリケーション・コード・ページをオーバーライドできる場合のサーバー上でのデータ	INGEST ユーティリティーがアプリケーション・コード・ページをオーバーライドできない場合のサーバー上でのデータ
CHAR	CHAR	X'E9'	X'82'	X'C3A9'	X'C3A9'
CHAR FOR BIT DATA	CHAR FOR BIT DATA	X'E9'	X'E9'	X'E9'	X'E9'
CHAR FOR BIT DATA	CHAR	X'E9'	X'E9'	**X'C3A9'**	**X'C39A'** ("Ú")
CHAR	CHAR FOR BIT DATA	X'E9'	X'82'	**X'E9'**	**X'82'**

fourth 列目のデータは、INGEST ユーティリティーがアプリケーション・コ

ード・ページをオーバーライドできる場合に DB2 に送るデータです。4 列目のデータは、 INGEST ユーティリティーが、アプリケーション・コード・ページをオーバーライドできない場合に送るデータです。フィールド定義と列定義の FOR BIT DATA 属性が異なる上記の表では、結果が異なっていることに注意してください。

コード・ページ・エラー

入力のコード・ページ、アプリケーションのコード・ページ、またはデータベースのコード・ページが異なる場合、INGEST ユーティリティー、DB2、またはその両方がコード・ページ変換を実行します。以下のケースのいずれかで、DB2 がコード・ページの変換をサポートしない場合、INGEST ユーティリティーはエラーを出し、コマンドは終了します。

変換が必要な場合	変換前	変換後	何が変換を実行するか
INGEST コマンドに、入力データのコード・ページに変換する必要のあるストリングまたは SQL ID が含まれている。	アプリケーションのコード・ページ	入力データのコード・ページ	INGEST ユーティリティー
ユーティリティーがアプリケーションのコード・ページを入力データのコード・ページにオーバーライドできる。	入力のコード・ページ	データベース・コード・ページ	DB2
ユーティリティーがアプリケーションのコード・ページをオーバーライドできない。	入力のコード・ページ	アプリケーションのコード・ページ	INGEST ユーティリティー
ユーティリティーがアプリケーションのコード・ページをオーバーライドできない。	アプリケーションのコード・ページ	データベース・コード・ページ	DB2

DB2 pureScale環境での INGEST 操作

DB2 pureScale環境で INGEST ユーティリティーを使用する場合、注意すべき追加の考慮事項がいくつかあります。

各フラッシャーが DB2 pureScale インスタンス上のデータベースに接続すると、次のいずれかになります。

- **SET CLIENT** コマンドが **CONNECT_MEMBER** オプション付きで発行された場合、フラッシャーはそのメンバーに接続する。
- それ以外の場合、フラッシャーは接続先のメンバーを指定しない。この場合、DB2 クライアントは接続レベルのワークロード・バランシング (WLB) を使用して接続先のメンバーを選択する。

INGEST ユーティリティーの多重呼び出しが同一メンバーに対して作動するか、異なるメンバーに対して作動するかは、**SET CLIENT** コマンドが **CONNECT_MEMBER** オプション付きで指定されているか、および WLB がどのメンバーを選択するかによって異なります。

また、接続先メンバーを明示的に指定しない場合は、すべてのメンバーがアクセスできる共有ディスク上に INGEST ファイルが配置されていることを確認してください。

パーティション・データベース環境での INGEST 操作

INGEST ユーティリティーを使用して、パーティション・データベース環境にデータを移動できます。

パーティション・データベースで **INGEST** コマンドを実行すると、**num_flushers_per_partition** 構成パラメーターの指定に従って、パーティションごとに 1 つ以上のフラッシャーが使用されます。デフォルトは以下のようになります。

```
max(1, ((number of logical CPUs)/2)/(number of partitions) )
```

このパラメーターは 0 に設定することもできます。この場合、全パーティションで 1 つのフラッシャーが使用されることになります。

各フラッシャーは、データの送信先となるパーティションに直接接続します。接続を成功させるには、すべての DB2 サーバー・パーティションが、同じポート番号を使用してクライアント接続を受信する必要があります。

ターゲット表が分散キーのあるタイプである場合、INGEST ユーティリティーは、次のようにして各レコードが属するパーティションを決定します。

1. すべての分散キーに、1 つだけ対応するフィールドまたは定数の値があるかどうか調べます。これに該当するのは、次の場合です。
   - INSERT ステートメントで、列のリストにすべての分散キーが含まれており、各分散キーに関する VALUES リスト内の対応する項目が、フィールド名または定数である。
   - UPDATE または DELETE ステートメントで、WHERE 述部が次の形式である。

     ```
     (dist-key-col1 = value1) AND (dist-key-col2 = value2) AND ...
     (dist-key-coln = valuen) [AND any-other-conditions]
     ```

     ここで *dist-keycol1* から *dist-key-coln* は、すべての分散キーであり、各 *value* はフィールド名または定数です。
   - MERGE ステートメントで、検索条件が、上記の UPDATE および DELETE の形式である。
2. すべての分散キーに、1 つだけ対応するフィールドまたは定数の値が存在する場合、 INGEST ユーティリティーは、その分散キーを使用してパーティション番号を特定し、そのパーティション用のフラッシャーのいずれかにレコードを転送します。

注: 次の場合、INGEST ユーティリティーはレコードのパーティションを特定しません。複数のフラッシャーが存在する場合、INGEST ユーティリティーは、ランダムに選択したフラッシャーにレコードを転送します。

- ターゲット表が分散キーのないタイプである。
- 列のリスト (INSERT) または述部 (UPDATE, MERGE, DELETE) が、全分散キーを指定していない。次の例では、キー列 2 から 8 が欠落しています。

```
UPDATE my_table SET data = $data
    WHERE (key1 = $key1) AND (key9 = $key9);
```

- 分散キーが、次の例のように、複数のフィールドまたは値に対応している。

```
UPDATE my_table SET data = $data
    WHERE key1 = $key11 OR key1 = $key12;
```

- 分散キーが、次の例のように、式に対応している。

```
INGEST FROM FILE ...
    INSERT INTO my_table(dist_key, col1, col2)
        VALUES($field1 + $field2, $col1, $col2);
```

- 分散キー列のタイプは DB2SECURITYLABEL です。
- 分散キーに対応するフィールドのタイプは数値タイプですが、分散キー列のタイプは異なる数値タイプであるか、あるいは異なる精度または位取りを持ちます。

サンプル INGEST ユーティリティー・スクリプト

INGEST ユーティリティーのサンプル・スクリプトを使用することにより、新しいファイルを処理するたびに新規の INGEST コマンドを記述する作業を自動化できます。

サンプル・スクリプト `ingest_files.sh` は、新規ファイルをチェックして、そのファイルを処理するための INGEST コマンドを生成する操作を自動化するシェル・スクリプトです。このスクリプトは、以下のタスクをこの順序で実行します。

1. ディレクトリーをチェックして、処理すべき新規ファイルがあるかどうか調べる。ファイルがないとスクリプトは終了する。

   注: このスクリプトは、指定されたディレクトリーに、取り込もうとしている表のためのファイルのみが含まれていることを想定しています。
2. 新規ファイルの名前を取得して、それぞれのファイル用に別個の INGEST コマンドを生成する。
3. INGEST コマンドを実行して、戻りコードを処理する。
4. 処理されたファイルを成功ディレクトリーまたは失敗ディレクトリーに移動する。

スクリプトは、インストール・ディレクトリー下の `samples/admin_scripts` ディレクトリーにあります。

環境に合わせたスクリプトの変更

`ingest_files.sh` スクリプトは、独自のスクリプトを作るためのベースとして使用することができます。重要な変更点としては、以下のものがあります。

- サンプル値 (データベース名、表の名前など) を独自の値に置き換える

- サンプルの INGEST コマンドを独自のコマンドに置き換える
- スクリプトで指定されているディレクトリーを作成する

スクリプトは、単一の表に取り込むためのデータを含むファイルを処理します。複数の表にデータを取り込むためには、取り込み先の表ごとにメカニズムを複製するか、複数の表を扱えるようにメカニズムを一般化します。

サンプル・シナリオ

新規 INGEST コマンドの生成を自動化するためにサンプル・スクリプトをユーザーのデータウェアハウスに適合させる方法を示すため、資料にはサンプル・シナリオが含まれています。

シナリオ: INGEST ユーティリティーを使用して、継続的に送られてくるファイルを処理する

次のシナリオは、絶え間なく継続的に送られてくるデータ・ファイルを自動的に INGEST するように、データウェアハウスを構成する方法を示しています。

問題: データウェアハウスによっては、1 日中絶え間なくファイルが到着し、到着したら直ちにそのファイルを処理する必要があります。つまり、新規ファイルが到着するたびに、その新規ファイルを処理するように指定した **INGEST** コマンドを実行する必要があります。

解決方法: 新規ファイルがあるかどうかのチェック、新しい **INGEST** コマンドの生成、およびそのコマンドの実行を自動的に行うスクリプトを作成します。`ingest_files.sh` は、このようなスクリプトのサンプルです。また、このシェル・スクリプトの実行頻度を指定するために、 crontab のエントリーを作成する必要もあります。

継続的に送られてくるファイルを処理するために、この仕組み (つまり、スクリプトおよび chrontab エントリー) を実装するには、次の前提条件および依存関係が満たされていなければなりません。

- ターゲット表がターゲット・データベースに作成されている
- INGEST ユーティリティーを使用する準備が整っている (つまり、クライアント・マシン上にインストールされ、セットアップされている)
- INGEST コマンドは指定されていて、また、テスト・ファイルを使用した手動実行によって検証されている
- **INGEST** コマンドで参照するオブジェクト (例外表など) が作成されている
- crontab ファイルが、INGEST ユーティリティーを実行するシステム上に作成されている
- 入力ファイルを作成し、それらのファイルを、スクリプトで使用するソース・ディレクトリーに移動させる権限がユーザーにある

1. ユーザーは、`ingest_files.sh` をテンプレートとして使用し、次のようにして新しいスクリプトを作成します。
   a. 次のサンプルの入力値を、ユーザーの値を反映するように置き換える
      - INPUT_FILES_DIRECTORY
      - DATABASE_NAME

- SCHEMA_NAME
- TABLE_NAME
- SCRIPT_PATH

b. サンプルの **INGEST** コマンドを置き換える

c. このスクリプトを `populate_table1_script` として保存する

2. crontab ファイルにエントリーを追加し、スクリプトの実行頻度を指定する。年間を通して毎日、1 分間隔で 1 日に 24 時間、スクリプトを実行するには、次の行を追加します。

```
1 * * * * $HOME/bin/populate_table1_script
```

3. 新規入力ファイルを作成してソース・ディレクトリーに追加することによって、このスクリプトのテストを行います。

その他のデータ移動オプション

ADMIN_MOVE_TABLE プロシージャーによるオンラインでの表の移動

ADMIN_MOVE_TABLE プロシージャーを使用すると、オンラインまたはオフラインでの移動を用いて表を移動できます。コスト、スペース、移動のパフォーマンス、およびトランザクションのオーバーヘッドと比べて、可用性の重要性の方がより大きい場合には、オフラインの表移動ではなく、オンラインの表移動を使用します。

始める前に

表および索引のコピー、ステージング表、および追加のログ項目に対応できる十分なディスク・スペースがあることを確認してください。

このタスクについて

表をオンラインのまま移動する場合は、ストアード・プロシージャーを 1 回だけ呼び出すことも、複数回 (プロシージャーが実行する操作ごとに 1 回ずつ) 呼び出すこともできます。複数回の呼び出しを行うと、移動のキャンセル、更新を行う際にターゲット表をオフラインにするタイミングの制御などの追加オプションを選択できます。

SYSPROC.ADMIN_MOVE_TABLE プロシージャーを呼び出すと、ソース表のシャドー・コピーが作成されます。このコピー・フェーズでは、トリガーを使用してソース表への変更 (更新、挿入、または削除) がキャプチャーされ、ステージング表に書き込まれます。コピー・フェーズが完了すると、ステージング表にキャプチャーされた変更内容がシャドー・コピーに再生されます。その後、ストアード・プロシージャーはソース表を短期間オフラインにして、ソース表名と索引名をシャドー・コピーとその索引に割り当てます。その後、シャドー表がソース表に置き換わってオンラインになります。デフォルトでは、ソース表はドロップされますが、KEEP オプションを使用すると、別の名前でそれを保持できます。

索引 (特にユニーク索引) がない表でオンラインの移動を実行することは避けてください。ユニーク索引がない表をオンライン移動するとデッドロックが生じる可能性があり、再生が複雑またはコストが高くなる場合があります。

手順

表をオンラインで移動するには、以下のようにします。

1. 以下のいずれかの方法で ADMIN_MOVE_TABLE プロシージャーを呼び出します。
   - ADMIN_MOVE_TABLE プロシージャーを一度呼び出します。その際、少なくともソース表のスキーマ名、ソース表名、および操作タイプとして MOVE を指定します。例えば、同じ表スペース内の既存の表にデータを移動するには、以下の構文を使用します。

     ```
     CALL SYSPROC.ADMIN_MOVE_TABLE (
     'schema name',
     'source table',
     '',
     '',
     '',
     '',
     '',
     '',
     '',
     '',
     'MOVE')
     ```

   - ADMIN_MOVE_TABLE プロシージャーを、操作ごとに一度ずつ、複数回呼び出します。その際、少なくともソース表のスキーマ名、ソース表名、および操作名を指定します。例えば、同じ表スペース内の新しい表にデータを移動するには、以下の構文を使用します。

     ```
     CALL SYSPROC.ADMIN_MOVE_TABLE (
     'schema name',
     'source table',
     '',
     '',
     '',
     '',
     '',
     '',
     '',
     '',
     'operation name')
     ```

     ここで、*operation name* は INIT、COPY、REPLAY、VERIFY、または SWAP のいずれかの値です。これらの操作の順序でプロシージャーを呼び出す必要があります。例えば、最初の呼び出しでは操作名として INIT を指定しなければなりません。

     注: VERIFY 操作はコストがかかります。この操作を実行するのは、表の移動のためにこの操作が必要な場合だけにしてください。
2. オンライン移動が失敗する場合、再実行してください。
   a. 表の移動が失敗する原因となる問題を修正します。
   b. 表移動が失敗した際に進行中だったステージを判別します。そのためには、SYSTOOLS.ADMIN_MOVE_TABLE プロトコル表で状況を照会します。

c. ストアード・プロシージャーを再び呼び出します。その際、適切なオプションを指定します。
   - プロシージャーの状況が INIT の場合、INIT オプションを使用します。
   - プロシージャーの状況が COPY の場合、COPY オプションを使用します。
   - プロシージャーの状況が REPLAY の場合、REPLAY または SWAP オプションを使用します。
   - プロシージャーの状況が CLEANUP の場合、CLEANUP オプションを使用します。

   オンラインの表移動の状況が COMPLETED または CLEANUP ではない場合、ストアード・プロシージャーに CANCEL オプションを指定すると移動をキャンセルできます。

例

例 1: スキーマ SVALENTI にある T1 表を、T1 をオフラインにすることなく ACCOUNTING 表スペースに移動させます。新しい表スペースに表を移動させるため、DATA、INDEX、および LONG の各表スペースを指定します。

```
CALL SYSPROC.ADMIN_MOVE_TABLE(
'SVALENTI',
'T1',
'ACCOUNTING',
'ACCOUNTING',
'ACCOUNTING',
'',
'',
'',
'',
'',
'',
'MOVE')
```

例 2: スキーマ EBABANI の T1 表を、T1 をオフラインにすることなく ACCOUNTING 表スペースに移動させ、移動後も元の表のコピーを保持します。COPY_USE_LOAD および LOAD_MSGPATH オプションを使用して、ロード・メッセージ・ファイル・パスを指定します。新しい表スペースに表を移動させるため、DATA、INDEX、および LONG の各表スペースを指定します。元の表は、'EBABANI'.'T1AAAAVxo' のような名前で保持されます。

```
CALL SYSPROC.ADMIN_MOVE_TABLE(
'EBABANI',
'T1',
'ACCOUNTING',
'ACCOUNTING',
'ACCOUNTING',
'',
'',
'',
'',
'KEEP, COPY_USE_LOAD,LOAD_MSGPATH "/home/ebabani"',
'MOVE')
```

例 3: T1 表を同じ表スペース内で移動させます。 T1 の列 C1 は、非推奨のデータ・タイプ LONG VARCHAR を使用しているので、互換性のあるデータ・タイプを使用するように変更します。

```
CALL SYSPROC.ADMIN_MOVE_TABLE(
'SVALENTI',
'T1',
'',
'',
'',
'',
'',
'',
'',
'C1 VARCHAR(1000), C2 INT(5), C3 CHAR(5), C4 CLOB',
'',
'MOVE')
```

注: この操作の際には、列名を変更することはできません。

例 4: 以下のステートメントによって作成された T1 表があります。

```
CREATE TABLE T1(C1 BIGINT,C2 BIGINT,C3 CHAR(20),C4 DEC(10,2),C5 TIMESTAMP,C6 BIGINT
 GENERATED ALWAYS AS (C1+c2),C7 GRAPHIC(10),C8 VARGRAPHIC(20),C9 XML
```

この表を、列 C5 および C6 をドロップして、同じ表スペース内で移動させます。

```
CALL SYSPROC.ADMIN_MOVE_TABLE(
'SVALENTI',
'T1',
'',
'',
'',
'',
'',
'',
'',
'c1 BIGINT,c2 BIGINT ,c3 CHAR(20),c4 DEC(10,2),c7 GRAPHIC(10),c8 VARGRAPHIC(20),c9 XML',
'',
'MOVE')
```

例 5: 表スペース TS1 および TS2 で 2 つの範囲が定義されている範囲パーティション表があります。表を表スペース TS3 に移動しますが、最初の範囲は TS1 に残します。

```
CREATE TABLE "EBABANI "."T1"  (
 "I1" INTEGER ,
 "I2" INTEGER )
  DISTRIBUTE BY HASH("I1")
 PARTITION BY RANGE("I1")
 (PART "PART0" STARTING(0) ENDING(100) IN "TS1",
 PART "PART1" STARTING(101) ENDING(MAXVALUE) IN "TS2");
```

スキーマ EBABANI の T1 表を、TS3 表スペースに移動します。パーティション定義を指定します。

```
DB2 "CALL SYSPROC.ADMIN_MOVE_TABLE
('EBABANI',
'T1',
'TS3',
'TS3',
'TS3',
'',
'',
'(I1) (STARTING 0 ENDING 100 IN TS1 INDEX IN TS1 LONG IN TS1,
 STARTING 101 ENDING MAXVALUE IN TS3 INDEX IN TS3 LONG IN TS3)',
'',
'',
'MOVE')"
```

IBM レプリケーション・ツールのコンポーネント

IBM は、Q レプリケーションおよび SQL レプリケーションという 2 つの主要なレプリケーション・ソリューションを提供しています。

Q レプリケーションの主要コンポーネントは、Q キャプチャー・プログラムと Q アプライ・プログラムです。 SQL レプリケーションの主要コンポーネントは、キャプチャー・プログラムとアプライ・プログラムです。両方のタイプのレプリケーションがレプリケーション・アラート・モニター・ツールを共有します。これらのレプリケーション・コンポーネントのセットアップと管理は、レプリケーション・センターおよび ASNCLP コマンド行プログラムを使って行えます。

以下のリストは、これらのレプリケーション・コンポーネントを簡潔に要約しています。

Q キャプチャー・プログラム

DB2 リカバリー・ログを読み取って DB2 ソース表に対する変更を収集し、コミットされたソース・データを WebSphere® MQ メッセージに変換します。このメッセージは、サブスクライブ・アプリケーションへの XML 形式として発行することもでき、Q アプライ・プログラムへのコンパクト形式としてレプリケーションすることもできます。

Q アプライ・プログラム

キューから WebSphere MQ メッセージを取り、メッセージを SQL ステートメントに変換し、ターゲット表またはストアード・プロシージャーを更新します。サポートされるターゲットとしては、DB2 データベースまたはサブシステム、および Oracle、Sybase、Informix®、Microsoft SQL Server のデータベース (フェデレーテッド・サーバーのニックネームを使ってアクセスする) があります。

キャプチャー・プログラム

DB2 リカバリー・ログを読み取って登録済みのソース表またはビューに対する変更を収集した後、コミットされたトランザクション・データを変更データ (CD) 表と呼ばれるリレーショナル表にステージングします。このデータはターゲット・システムでのコピー準備が整うまでこの表に保管されます。また、SQL レプリケーションは、キャプチャー・トリガーも提供します。これは、整合変更データ (CCD) 表と呼ばれるステージング表に、非 DB2 ソース表に対する変更のレコードを取り込むものです。

アプライ・プログラム

ステージング表からデータを読み取り、ターゲットに対して適切な変更を加えます。非 DB2 データ・ソースの場合、アプライ・プログラムはフェデレーテッド・データベース上の表のニックネームを使って CCD 表を読み取り、ターゲット表に対して適切な変更を加えます。

レプリケーション・アラート・モニター

Q キャプチャー、Q アプライ、キャプチャー、およびアプライ・プログラムの正常性をチェックするユーティリティー。このモニターはさまざまな状況をチェックし

ます。例えば、プログラムの終了、警告またはエラー・メッセージの発行、指定された値のしきい値の到達、または特定のアクションの実行などの状況です。これらをチェックした後、E メール・サーバー、ページャー、または z/OS® コンソールに通知を発行します。

レプリケーション・センターは、以下を行うために使用します。

- ソース表の登録、サブスクリプション、イベント発行、キュー・マップ、アラート条件、およびその他のオブジェクトの定義。
- レプリケーション・プログラムの開始、停止、中断、再開、および再初期化。
- 自動コピーの時間指定。
- データの SQL 拡張の指定。
- ソース表とターゲット表のリレーションシップの定義。

スキーマのコピー

db2move ユーティリティーおよび ADMIN_COPY_SCHEMA プロシージャーにより、データベース・スキーマのコピーを迅速に作成できます。モデル・スキーマを確立すると、新しいバージョンを作成するためのテンプレートとしてそれを使用できます。

手順

- 同じデータベース内の単一のスキーマをコピーするには、ADMIN_COPY_SCHEMA プロシージャーを使用します。
- 単一スキーマまたは複数のスキーマをソース・データベースからターゲット・データベースにコピーするには、**-co COPY** アクションを指定して **db2move** ユーティリティーを使用します。 ソース・スキーマからのほとんどのデータベース・オブジェクトは、新しいスキーマの下のターゲット・データベースにコピーされます。

トラブルシューティングのヒント

ADMIN_COPY_SCHEMA プロシージャーと **db2move** ユーティリティーはどちらも **LOAD** コマンドを呼び出します。ロードの処理中、データベース・ターゲット・オブジェクトのある表スペースはバックアップ・ペンディング状態になります。

ADMIN_COPY_SCHEMA プロシージャー

前述の注で説明したように、COPYNO オプションを指定したプロシージャーを使用すると、ターゲット・オブジェクトがある表スペースがバックアップ・ペンディング状態になります。表スペースの SET INTEGRITY ペンディング状態を解除するために、このプロシージャーは SET INTEGRITY ステートメントを発行します。ターゲット表オブジェクトに参照制約が定義されている状態では、ターゲット表も SET INTEGRITY ペンディング状態になります。表スペースは既にバックアップ・ペンディング状態であるため、ADMIN_COPY_SCHEMA プロシージャーが SET INTEGRITY ステートメントを発行しようとしても失敗します。

この状態を解決するには、**BACKUP DATABASE** コマンドを実行して、影響を受ける表スペースのバックアップ・ペンディング状態を解除します。次いで、このプロシージャーによって生成されたエラー表の Statement_text 列を参照し、SET INTEGRITY ペンディング状態にある表のリストを見つけます。

次に、リストされている各表に SET INTEGRITY ステートメントを発行し、各表の SET INTEGRITY ペンディング状態を解除します。

db2move ユーティリティー

このユーティリティーは、以下のタイプを除いて、許容されるスキーマ・オブジェクトをすべてコピーしようとします。

- 表階層
- ステージング表 (複数パーティション・データベース環境でのロード・ユーティリティーではサポートされません)
- JAR (Java™ ルーチン・アーカイブ)
- ニックネーム
- パッケージ
- ビュー階層
- オブジェクト特権 (新規オブジェクトはすべてデフォルト許可で作成されます。)
- 統計 (新規オブジェクトに統計情報は含まれません。)
- 索引拡張 (ユーザー定義構造化タイプに関連)
- ユーザー定義構造化タイプおよびそのトランスフォーム関数

サポートされないタイプに関するエラー

サポートされないタイプのいずれかのオブジェクトがソース・スキーマで検出されると、項目がエラー・ファイルに記録されます。そのエラー・ファイルには、サポートされないオブジェクト・タイプが検出されたことが示されます。ただし COPY 操作は成功します。ログに記録された項目は、この操作でコピーされなかったオブジェクトを通知するためのものです。

スキーマを伴わないオブジェクト

表スペースなどの、スキーマを伴わないオブジェクト、およびイベント・モニターは、スキーマのコピー操作時には操作されません。これらは、スキーマのコピー操作が呼び出される前に、ターゲット・データベース上で作成する必要があります。

複製された表

複製された表をコピーする場合、表の新規コピーはレプリケーションには使用できません。表は通常表として再作成されます。

異なるインスタンス

ソース・データベースは、ターゲット・データベースと同じインスタンス内にない場合は、カタログされる必要があります。

SCHEMA_MAP オプション

SCHEMA_MAP オプションを使用して、ターゲット・データベース上で異なるスキーマ名を指定する場合、元のスキーマ名を新しいスキーマ名に置き換えるために、スキーマのコピー操作はオブジェクト定義ステートメントの最小限の構文解析のみを実行します。例えば、SQL プロシージャーの内容の中に表示されるオリジナル・スキーマのインスタンスは新しいスキーマ名に置き換えられません。したがって、スキーマのコピー操作はそれらのオブジェクトを再作成できない場合があります。ステージング表、結果表、マテ

リアライズ照会表などの場合も同様です。コピー操作の完了後に、エラー・ファイル内の DDL を使用して、それらの失敗したオブジェクトを手動で再作成することができます。

オブジェクト間の相互依存

スキーマのコピー操作は、それらのオブジェクト間の相互依存性を満たす順序でオブジェクトを再作成しようとします。例えば、表 T1 にユーザー定義関数 U1 を参照する列が含まれる場合、T1 を再作成する前に U1 を再作成します。ただし、プロシージャーの従属情報がカタログ内で前もって使用可能なわけではないため、プロシージャーを再作成する際に、スキーマのコピー操作では最初にすべてのプロシージャーを再作成し、それから失敗したものを再作成しようとします (それらのプロシージャーが前の試行時に正常に作成されたプロシージャーに依存していれば、それ以降の試行時には正常に再作成されると想定しています)。それ以降の試行時に 1 つ以上のプロシージャーを正常に再作成できる限り、この操作は失敗したプロシージャーを引き続き再作成しようとします。プロシージャーの再作成試行時に、エラー (および DDL) がエラー・ファイルに毎回記録されます。エラー・ファイルに同じプロシージャーの項目が多数表示されることがありますが、それらのプロシージャーはそれ以降の試行時に正常に再作成されている可能性がありますスキーマのコピー操作の完了時に SYSCAT.PROCEDURES 表を照会して、エラー・ファイルにリストされているそれらのプロシージャーが正常に再作成されたかどうか判断する必要があります。

詳細については、ADMIN_COPY_SCHEMA プロシージャーおよび **db2move** ユーティリティーを参照してください。

db2move ユーティリティーを使用したスキーマ・コピーの例

1 つまたは複数のスキーマをソース・データベースからターゲット・データベースにコピーするには、**-co COPY** アクションを指定して **db2move** ユーティリティーを使用します。モデル・スキーマを確立すると、新しいバージョンを作成するためのテンプレートとしてそれを使用できます。

例 1: -c COPY オプションの使用

-co COPY オプションを指定した以下の **db2move** の例は、スキーマ BAR をサンプル・データベースからターゲット・データベースにコピーして、それを FOO に名前変更します。

```
db2move sample COPY -sn BAR -co target_db  target schema_map
  "((BAR,FOO))" -u userid -p password
```

新規の (ターゲット) スキーマ・オブジェクトは、ソース・スキーマのオブジェクトと同じオブジェクト名を使用して作成されますが、修飾子はターゲット・スキーマのものが使用されます。表のコピーを作成する際には、ソース表のデータを共にコピーすることも、しないこともできます。ソース・データベースおよびターゲット・データベースは、別々のシステムに存在できます。

例 2: COPY 操作中に表スペース名のマッピングを指定する

以下の例は、**db2move COPY** 操作時にソース・システムからの表スペースの代わりに使用される、特定の表スペース名マッピングを指定する方法を示しています。 SYS_ANY キーワードを指定して、ターゲット表スペースを、

デフォルトの表スペース選択アルゴリズムを使用して選択することを指示できます。この場合、**db2move** ユーティリティーは、ターゲットとして使用可能な任意の表スペースを選択します。

```
db2move sample COPY -sn BAR -co target_db  target schema_map
  "((BAR,FOO))" tablespace_map "(SYS_ANY)" -u userid -p password
```

SYS_ANY キーワードは、すべての表スペースに使用できます。または、一部の表スペースに固有のマッピングを指定し、残りにはデフォルトの表スペース選択アルゴリズムを指定できます。

```
db2move sample COPY -sn BAR -co target_db  target schema_map "
  ((BAR,FOO))" tablespace_map "((TS1, TS2),(TS3, TS4), SYS_ANY)"
  -u userid -p password
```

これは、表スペース TS1 が TS2 にマップされ、TS3 が TS4 にマップされ、残りの表スペースがデフォルトの表スペース選択アルゴリズムを使用することを示しています。

例 3: COPY 操作後のオブジェクト所有者の変更

正常に COPY を実行した後に、ターゲット・スキーマで作成された新しい各オブジェクトの所有者を変更することができます。ターゲット・オブジェクトのデフォルト所有者は接続ユーザーです。以下のようなオプションを指定すると、新しい所有者に所有権が移転します。

```
db2move sample COPY -sn BAR -co target_db  target schema_map
  "((BAR,FOO))" tablespace_map "(SYS_ANY)" owner jrichards
  -u userid -p password
```

ターゲット・オブジェクトの新規所有者は jrichards です。

db2move ユーティリティーは、ソース・スキーマとターゲット・スキーマが別々のシステムにある場合にはターゲット・システムで開始する必要があります。あるデータベースから別のデータベースにスキーマをコピーする場合、このアクションを行うには、ソース・データベースからコピーされるスキーマ名のリスト (コンマで区切られたもの) およびターゲット・データベース名が必要です。

スキーマをコピーするには、オペレーティング・システムのコマンド・プロンプトから **db2move** を次のように発行します。

```
db2move dbname COPY -co COPY-options
 -u userid -p password
```

db2move - データベース移動ツール

このツールを EXPORT、IMPORT、LOAD いずれかのモードで使用すると、ワークステーション上にある DB2 データベース間で、大量の表を簡単に移動することができます。 COPY モードで使用すると、このツールによってスキーマの複写が容易になります。

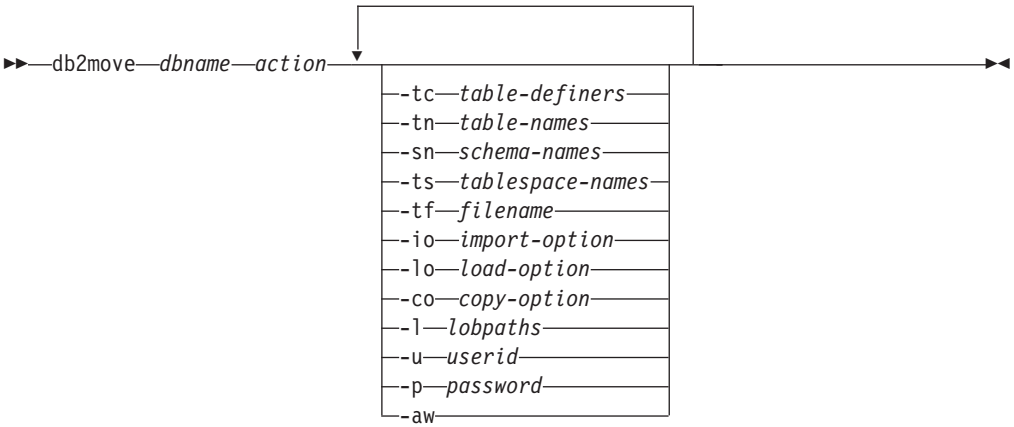
また、特定のデータベースのシステム・カタログ表を照会し、すべてのユーザー表のリストをコンパイルします。そして、これらの表を PC/IXF フォーマットでエクスポートします。 PC/IXF ファイルは、同じシステム上の別のローカル DB2 データベースにインポートまたはロードするか、または別のワークステーション・プラ

ットフォームに転送し、そのプラットフォームで DB2 データベースにインポートまたはロードすることができます。構造化タイプ列がある表は、このツールを使用しても移動しません。

許可

このツールは、ユーザーから要求されるアクションにしたがって、DB2 エクスポート、インポート、およびロード API を呼び出します。したがって、要求元ユーザー ID には、これらの API に求められる正しい権限がなければなりません。この権限がないと、要求は失敗します。

コマンド構文



コマンド・パラメーター

dbname
データベースの名前。

action 以下のうちの 1 つでなければなりません。

EXPORT
options のフィルター基準を満たすすべての表をエクスポートします。 options の指定がない場合には、すべての表をエクスポートします。内部ステージング情報は db2move.lst ファイルに保管されます。

IMPORT
内部ステージング・ファイル db2move.lst にリストされているすべての表をインポートします。 IMPORT の特定のアクションには、**-io** オプションを使用します。

LOAD 内部ステージング・ファイル db2move.lst にリストされているすべての表をロードします。 LOAD の特定のアクションには、**-lo** オプションを使用します。

COPY ターゲット・データベースにスキーマを複製します。ターゲット・データベースはローカル・データベースでなければなりません。**-sn** オプションを使用して、1 つ以上のスキーマを指定します。 **COPY** 特定オプションの -co オプションを参照してください。**-tn** または

-tf オプションを使用して、LOAD_ONLY モードの表をフィルターします。ADMIN_COPY_SCHEMA() ストアード・プロシージャーを使用する場合も、**-COPY** オプションを指定した **db2move** ユーティリティーを使用する場合も、SYSTOOLSPACE という名前の表スペースを使用する必要があります。

各アクション中に生成されるファイルのリストについては、下記のセクションを参照してください。

-tc *table-definers*

デフォルトはすべての定義者です。

これは EXPORT アクションのみです。指定されると、このオプションでリストされる定義者が作成する表のみがエクスポートされます。指定されない場合、デフォルトではすべての定義者を使用します。複数の定義者を指定する場合、それぞれをコンマで区切る必要があります。定義者 ID 間にブランクを入れることはできません。このオプションを **-tn** *table-names* オプションとともに使用すると、エクスポートする表を選択できます。

アスタリスク (*) は、ストリング中のどこにでも入れられるワイルドカード文字として使用できます。

-tn *table-names*

デフォルトはすべてのユーザー表です。

これは EXPORT または COPY アクションのみです。

EXPORT アクションで指定されると、指定されたストリング内の表と名前が一致する表だけがエクスポートされます。指定されない場合、デフォルトではすべてのユーザー表を使用します。複数の表名を指定する場合、それぞれをコンマで区切る必要があります。表名間にブランクを入れることはできません。スキーマをフィルター操作するには、修飾なしで表名をリストし、**-sn** オプションを使用する必要があります。

エクスポートの場合、アスタリスク (*) は、ストリング中のどこにでも入れられるワイルドカード文字として使用できます。

COPY アクションで指定する場合、これに加えて **-co "MODE" LOAD_ONLY** *copy-option* も指定する必要があります。指定された表のデータだけがターゲット・データベースで再挿入されます。表名は、スキーマ修飾子と共に "schema"."table" の形式でリストします。

-sn *schema-names*

EXPORT のデフォルトは全スキーマです (COPY ではない)。

これが指定されると、一致するスキーマ名の表だけがエクスポートまたはコピーされます。複数のスキーマ名を指定する場合は、それぞれの名前をコンマで区切る必要があります。複数のスキーマ名の間にブランクを使用することはできません。 8 文字より短いスキーマ名は、8 文字の長さになるまで埋め込まれます。

エクスポートの場合: スキーマ名の部分にアスタリスク・ワイルドカード文字 (*) が使用された場合は、それがパーセント記号 (%) に変更され、WHERE 節の LIKE 述部にパーセント付きの表名が使用されます。指定されない場合、デフォルトではすべてのスキーマを使用します。 **-tn** または

-tc オプションと合わせて使用する場合、**db2move** は、スキーマが指定されたスキーマ名と一致し、定義者が指定された定義者と一致する表に対してのみ実行されます。 fred のようなスキーマ名の場合、アスタリスクを使用するときは、-sn fr*d ではなく -sn fr*d* のような指定が必要になります。

-ts *tablespace-names*
デフォルトはすべての表スペースです。

これは EXPORT アクションのみです。このオプションが指定されると、指定した表スペースにある表だけがエクスポートされます。表スペース名の部分にアスタリスク・ワイルドカード文字 (*) が使用された場合は、それがパーセント記号 (%) に変更され、WHERE 節の LIKE 述部にパーセント付きの表名が使用されます。 **-ts** オプションが指定されない場合、デフォルトではすべての表スペースを使用します。複数の表スペース名を指定する場合は、それぞれの名前をコンマで区切る必要があります。複数の表スペース名の間にブランクを使用することはできません。 8 文字より短い表スペース名は、8 文字の長さになるまで埋め込まれます。例えば、mytb のような表スペース名の場合、アスタリスクを使用するときは、-sn my*b ではなく -ts my*b* のような指定が必要になります。

-tf *filename*

EXPORT アクションで指定されると、指定されたファイル内の名前と正確に一致する名前の表だけがエクスポートされます。指定されない場合、デフォルトではすべてのユーザー表を使用します。表は 1 行に 1 つずつリストする必要があり、各表は完全に修飾する必要があります。ストリング内には、ワイルドカード文字を使用できません。 以下は、ファイルの内容の例です。

```
"SCHEMA1"."TABLE NAME1"
"SCHEMA NAME77"."TABLE155"
```

COPY アクションで指定する場合、これに加えて **-co "MODE" LOAD_ONLY** *copy-option* も指定する必要があります。ファイル内に指定された表のデータだけが、ターゲット・データベースで再挿入されます。表名は、スキーマ修飾子と共に "schema"."table" の形式でリストします。

-io *import-option*
デフォルトは REPLACE_CREATE です。インポート作成機能の制限については、『IMPORT コマンドの推奨されなくなったオプション CREATE および REPLACE_CREATE』を参照してください。

有効なオプションは、INSERT、INSERT_UPDATE、REPLACE、CREATE、および REPLACE_CREATE です。

-lo *load-option*
デフォルトは INSERT です。

有効なオプションは、INSERT および REPLACE です。

-co **db2move** アクションが COPY である場合、以下の **-co** 追加オプションを使用できます。

"TARGET_DB *db name* **[USER** *userid* **USING** *password***]"**
ユーザーがターゲット・データベースの名前、ユーザー ID、およびパスワードを指定できるようにします。 (ソース・データベースの

ユーザー ID およびパスワードは、既存の **-p** および **-u** オプションを使用して指定できます。) **USER USING** 節はオプションです。**USER** が userid を指定する場合は、**USING** 節の後にパスワードを指定します。パスワードが指定されない場合、**db2move** はパスワード情報を求めるプロンプトを出します。プロンプトが出されるのは、下記のセクションで説明するセキュリティー上の理由によります。**TARGET_DB** は COPY アクションには必須のオプションです。**TARGET_DB** は、ソース・データベースと同じにすることはできませんし、ローカル・データベースでなければなりません。同じデータベース内のスキーマをコピーするには、ADMIN_COPY_SCHEMA プロシージャーを使用できます。COPY アクションには、少なくとも 1 つのスキーマ (**-sn**) または 1 つの表 (**-tn** または **-tf**) の入力が必要です。

複数の **db2move** コマンドを実行してスキーマを 1 つのデータベースから別のデータベースにコピーすると、デッドロックになります。一度に 1 つのみの **db2move** コマンドを発行してください。コピー処理中にソース・スキーマ内の表を変更すると、ターゲット・スキーマのデータがコピー後に同一のものにならないことがあります。

"MODE"

DDL_AND_LOAD
ソース・スキーマの、すべてのサポートされるオブジェクトを作成し、ソース表データを表に追加します。これはデフォルト・オプションです。

DDL_ONLY
ソース・スキーマの、すべてのサポートされるオブジェクトを作成しますが、表にデータを再設定しません。

LOAD_ONLY
指定されたすべての表をソース・データベースからターゲット・データベースへロードします。表はターゲットに既に存在していなければなりません。LOAD_ONLY モードでは、**-tn** または **-tf** オプションを使って少なくとも 1 つの表を入力する必要があります。

これは、COPY アクションでのみ使用される任意指定のオプションです。

"SCHEMA_MAP"
ターゲットへコピーするときにユーザーがスキーマをリネームできるようにします。ソース・ターゲット間のスキーマ・マッピングをコンマで区切り、大括弧で囲んだリストを提供します。例えば、schema_map ((s1, t1), (s2, t2)) のようになります。これは、スキーマ s1 からのオブジェクトはターゲットのスキーマ t1 にコピーされ、スキーマ s2 からのオブジェクトはターゲットのスキーマ t2 へコピーされることを意味します。ターゲット・スキーマ名がソース・スキーマ名であるのがデフォルトで、推奨されています。この理由は、**db2move** がオブジェクト本体内に修飾オブジェクトのあるスキーマを変更しないことにあります。したがって、異なるター

ゲット・スキーマ名を使用すると、オブジェクト本体内に修飾オブジェクトがある場合に問題が生じるおそれがあります。

例えば、create view FOO.v1 as 'select c1 from FOO.t1'

この場合、スキーマ FOO の BAR へのコピー、v1 は以下のように再生成されます。create view BAR.v1 as 'select c1 from FOO.t1'

これは、スキーマ FOO がターゲット・データベースに存在しないため失敗するか、または FOO が BAR と異なるために予期しない結果になります。ソースと同じスキーマ名を保つことにより、これらの問題を避けることができます。スキーマ間に相互従属関係がある場合、すべての相互に従属するスキーマがコピーされなければなりません。あるいは、相互従属関係のあるオブジェクトのコピーでエラーになります。

例えば、create view FOO.v1 as 'select c1 from BAR.t1'

この場合、v1 のコピーは BAR がコピーされない場合に失敗するか、または、ターゲットの BAR がソースからの BAR と異なる場合、予期しない結果になります。**db2move** はスキーマの相互従属関係を検出しようとはしません。

これは、COPY アクションでのみ使用される任意指定のオプションです。

ターゲット・スキーマが既に存在する場合には、ユーティリティーは失敗します。 ADMIN_DROP_SCHEMA プロシージャーを使用して、スキーマと、そのスキーマに関連付けられたすべてのオブジェクトをドロップしてください。

"NONRECOVERABLE"

このオプションにより、ユーザーはロードのデフォルト動作をオーバーライドし、ロードが COPY-NO で行われるようにすることができます。デフォルトの動作では、ユーザーはロードされる各表スペースをバックアップするよう強制されます。この **NONRECOVERABLE** キーワードを指定すると、ユーザーは表スペースをバックアップするように即時に強制されることがありません。ただし、新しく作成された表が正しくリカバリーできるように、できるだけ早くバックアップを取ることを強くお勧めします。これは、COPY アクションで使用できる任意指定のオプションです。

"OWNER"

正常にコピーした後、ターゲット・スキーマに作成された各新規オブジェクトの所有者をユーザーが変更できるようにします。ターゲット・オブジェクトのデフォルト所有者は接続ユーザーになりますが、このオプションが指定された場合、所有権は新規所有者に移されます。これは、COPY アクションで使用できる任意指定のオプションです。

"TABLESPACE_MAP"

ユーザーは、コピー中に使用する表スペース名のマッピングを、ソース・システムの表スペースの代わりに指定できます。これは、大括弧で囲まれた表スペース・マッピングが配列されたものです。例

えば、tablespace_map ((TS1, TS2),(TS3, TS4)) のようにします。これは、表スペース TS1 からのすべてのオブジェクトはターゲット・データベースの表スペース TS2 にコピーされ、表スペース TS3 からのオブジェクトはターゲットの表スペース TS4 へコピーされることを意味します。 ((T1, T2),(T2, T3)) の場合、ソース・データベースの T1 にあるすべてのオブジェクトはターゲット・データベースの T2 に再作成され、ソース・データベースの T2 にあるどのオブジェクトもターゲット・データベースの T3 に再作成されることになります。デフォルトでは、ソースの表スペース名と同じ表スペース名を使用しますが、その場合には、表スペースのマッピング入力は必要ありません。指定された表スペースが存在しない場合、その表スペースを使用したオブジェクトのコピーは失敗し、エラー・ファイルにログされます。

ユーザーには、SYS_ANY キーワードを使用して、ターゲット表スペースの選択にデフォルトの表スペース選択アルゴリズムの使用を指定するオプションもあります。この場合、**db2move** は使用できる表スペースをどれでもターゲットとしての使用に選択することができます。SYS_ANY キーワードはすべての表スペースに対して使用できます。例えば、tablespace_map SYS_ANY とします。さらに、ユーザーは特定のマッピングを表スペースのいくつかに指定し、残りにデフォルトの表スペース選択アルゴリズムを指定することもできます。例えば、tablespace_map ((TS1, TS2),(TS3, TS4), SYS_ANY) のようにします。これは、表スペース TS1 は TS2 に、TS3 は TS4 にマップされるが、残った表スペースはデフォルトの表スペース・ターゲットを使用することを意味します。「SYS」で始まる表スペースはあり得ないため、SYS_ANY キーワードが使用されます。

これは、COPY アクションで使用できる任意指定のオプションです。

"PARALLEL" *number-of-threads*

このオプションを指定すると、スキーマ内の表のロード操作がいくつかのスレッドに分散されます。*number-of-threads* の値の範囲は 0 から 16 です。

- PARALLEL を指定しない場合、スレッドは使用されず、ロード操作は逐次実行されます。
- スレッド数を設定しないで PARALLEL を指定した場合、**db2move** ユーティリティーによって適切な値が選択されます。
- *number-of-threads* を設定して PARALLEL を指定した場合、指定した数のスレッドが使用されます。*number-of-threads* が 0 または 1 である場合、ロード操作は逐次実行されます。
- *number-of-threads* に指定できる最大値は 16 です。

これは、COPY アクションで使用できる任意指定のオプションです。

-l *lobpaths*

IMPORT および EXPORT の場合、このオプションが指定されると、これは XML パスにも使用されます。デフォルトは、現行ディレクトリーです。

このオプションは、LOB または XML ファイルが (EXPORT の一部として) 作成されるか、または (IMPORT または LOAD の一部として) 検索される絶対

パス名を指定します。複数のパスを指定する場合、それぞれをコンマで区切る必要があります。パス間にブランクを入れることはできません。複数のパスが指定された場合、EXPORT はラウンドロビン方式でそれらを使用します。つまり、1 つの LOB 文書を最初のパスに書き込み、それから 2 番目のパスに、という順に最後まで書き込み、その後最初のパスに戻ります。XML 文書でも同じです。最初のパスでファイルが見つからない場合 (IMPORT または LOAD 中)、2 番目のパスが使用される、という方法でパスが使用されます。

-u *userid*

デフォルトはログオン・ユーザー ID です。

ユーザー ID とパスワードはどちらも任意指定です。しかし、一方を指定した場合、他方も必ず指定する必要があります。コマンドがリモート・サーバーに接続するクライアント上で実行される場合、ユーザー ID とパスワードを指定する必要があります。

-p *password*

デフォルトはログオン・パスワードです。ユーザー ID とパスワードはどちらも任意指定です。しかし、一方を指定した場合、他方も必ず指定する必要があります。 **-p** オプションが指定されてもパスワードが指定されていない場合、**db2move** はパスワードを求めるプロンプトを出します。これは、セキュリティーの理由によります。コマンド行にパスワードを入力するとセキュリティー問題が生じます。例えば、**ps -ef** コマンドがパスワードを表示します。しかし、**db2move** がスクリプトを通して呼び出される場合は、パスワードを供給する必要があります。コマンドがリモート・サーバーに接続するクライアント上で発行される場合、ユーザー ID とパスワードを指定する必要があります。

-aw 警告を許します。 **-aw** が指定されていない場合、エクスポート中に警告があった表は db2move.lst ファイルに組み込まれません (表の .ixf ファイルと .msg ファイルが生成されていても)。しかし、あるシナリオ (データ切り捨てなど) では、そのように警告があった表でも db2move.lst ファイルに組み込んでしまいたい場合があります。そのようなとき、このオプションを指定すると、エクスポート中に警告を受け取った表を .lst ファイルに組み込むことができます。

例

- SAMPLE データベースのすべての表をエクスポートするには (すべてのオプションにデフォルト値を使用)、以下を発行します。

```
db2move sample export
```

- userid1 または us%rid2 のようなユーザー ID で作成され、tbname1 という名前、または %tbname2 のような表名を持つすべての表をエクスポートするには、以下を発行します。

```
db2move sample export -tc userid1,us*rid2 -tn tbname1,*tbname2
```

- すべての表を SAMPLE データベースにインポートするには、以下を発行します (LOB パス D:¥LOBPATH1 および C:¥LOBPATH2 で LOB ファイルが検索されます。この例は、Windows オペレーティング・システムにのみ該当します)。

```
db2move sample import -l D:¥LOBPATH1,C:¥LOBPATH2
```

- SAMPLE データベースのすべての表をロードするには、以下を発行します。(/home/userid/lobpath サブディレクトリーと tmp サブディレクトリーで、LOB ファイルが検索されます。この例は Linux および UNIX システムにのみ該当します。

```
db2move sample load -l /home/userid/lobpath,/tmp
```

- SAMPLE データベースのすべての表を、指定されたユーザー ID およびパスワードを使用して REPLACE モードでインポートするには、以下を発行します。

```
db2move sample import -io replace -u userid -p password
```

- スキーマ schema1 をソース・データベース dbsrc からターゲット・データベース dbtgt へ複写するには、以下を発行します。

```
db2move dbsrc COPY -sn schema1 -co TARGET_DB dbtgt USER myuser1 USING mypass1
```

- スキーマ schema1 をソース・データベース dbsrc からターゲット・データベース dbtgt へ複写し、そのターゲット上でスキーマを newschema1 に名前変更し、ソース表スペース ts1 をターゲットの ts2 へマップするには、以下を発行します。

```
db2move dbsrc COPY -sn schema1 -co TARGET_DB dbtgt USER myuser1 USING mypass1
     SCHEMA_MAP ((schema1,newschema1)) TABLESPACE_MAP ((ts1,ts2), SYS_ANY))
```

使用上の注意

- 1 つ以上のスキーマをターゲット・データベースにコピーする場合、各スキーマは、互いに依存しないものである必要があります。そうでない場合、オブジェクトの一部がターゲット・データベースに正常にコピーされない可能性があります。
- XML 列を含む表へのデータのロードは、 **LOAD** 操作でのみサポートされており、**COPY** 操作ではサポートされていません。回避策は、手動で **IMPORT** または **EXPORT** コマンドを実行するか、**db2move Export** および **db2move Import** 動作を使用することです。それらの表に GENERATED ALWAYS ID 列も含まれている場合は、表にデータをインポートできません。
- **db2move EXPORT** に続いて **db2move IMPORT** または **db2move LOAD** を指定すると、簡単に表データを移動できます。表に関連した他のすべてのデータベース・オブジェクト (別名、ビュー、トリガーなど)、およびこれらの表が依存するオブジェクト (ユーザー定義タイプ、ユーザー定義関数など) を手動で移動する必要があります。
- **CREATE** または **REPLACE_CREATE** オプションを指定した **IMPORT** アクションを使ってターゲット・データベース上に表を作成する場合 (どちらのオプションも非推奨になり、今後のリリースで除去される可能性があります)、『インポート済みの表の再作成』で説明されている制約が適用されます。 **REPLACE_CREATE** オプションの使用時の **db2move** インポート・フェーズ中に想定外のエラーが生じた場合、該当する tabnnn.msg メッセージ・ファイルを調べて、表の作成に対する制限事項が原因でエラーが起きたかどうかを確かめてください。
- **db2move** を使用して、ID 列 GENERATED ALWAYS を含む表をインポートまたはロードすることはできません。ただし、手動でこれらの表をインポートまたはロードすることは可能です。詳しくは、『ID 列のロードに関する考慮事項』または『ID 列のインポートに関する考慮事項』を参照してください。

- エクスポート、インポート、またはロード API が **db2move** によって呼び出されると、**FileTypeMod** パラメーターが lobsinfile に設定されます。つまり、LOB データが各表に対して、PC/IXF ファイルとは別のファイルに保持されます。
- **LOAD** コマンドは、データベースおよびデータ・ファイルが常駐するマシンでローカルに実行する必要があります。
- **db2move LOAD** を使用する場合、データベースの **logarchmeth1** が使用可能 (データベースがリカバリー可能) であれば、次のようになります。
  - **NONRECOVERABLE** オプションが指定されない場合、**db2move** はデフォルトの **COPY NO** オプションを使って db2Load API を呼び出します。ロードされた表が格納される表スペースは、ユーティリティー完了時にバックアップ・ペンディング状態に置かれます (表スペースをバックアップ・ペンディング状態から解除するには、データベース全体または表スペース全体のバックアップが必要です)。
  - **NONRECOVERABLE** オプションが指定されている場合、表スペースはバックアップ・ペンディング状態に置かれませんが、ロールフォワード・リカバリーが後で実行された場合、表はアクセス不能とマーク付けられるため、表をドロップする必要があります。ロード・リカバリー可能性オプションの詳細については、『ロードのパフォーマンスを改善するためのオプション』を参照してください。
- **IMPORT** または **LOAD** アクションを使用する **db2move** コマンドのパフォーマンスは、デフォルトのバッファー・プール IBMDEFAULTBP を変更し、構成パラメーター **sortheap**、**util_heap_sz**、**logfilsiz**、および **logprimary** を更新することによって、改善できます。
- **export** や **db2move** などのデータ移動ユーティリティーを実行する際に、照会コンパイラーが、基礎照会を基本表よりも MQT に対して実行する方が効率が良いと判別することがあります。そのような場合、照会は据え置きリフレッシュの MQT に対して実行され、ユーティリティーの結果は基礎表内のデータを正確に表していない可能性があります。
- **db2move** コマンドは、DB2 クライアントでは使用できません。クライアント・マシンから **db2move** コマンドを発行すると、エラー・メッセージ「db2move は、内部コマンドまたは外部コマンド、 操作可能なプログラムまたはバッチ ファイルとして認識されていません。」を受け取ります。この問題を避けるために、サーバー上で直接 **db2move** コマンドを発行できます。

EXPORT 使用時に必要とされるファイル/生成されるファイル

- 入力: なし。
- 出力:

  EXPORT.out
  EXPORT アクションの結果の要約。

  db2move.lst
  オリジナル表名のリスト、その対応する PC/IXF ファイル名 (tabnnn.ixf)、およびメッセージ・ファイル名 (tabnnn.msg)。このリスト、エクスポートされた PC/IXF ファイル、および LOB ファイル (tabnnnc.yyy) は、**db2move** IMPORT または LOAD アクションへの入力として使用されます。

tab*nnn*.ixf
特定の表の、エクスポートされる PC/IXF ファイル。

tab*nnn*.msg
対応する表のエクスポート・メッセージ・ファイル。

tab*nnnc*.*yyy*
特定の表の、エクスポートされる LOB ファイル。

nnn は表番号です。*c* はアルファベットの文字です。*yyy* は 001 から 999 の範囲内の数値です。

これらのファイルは、エクスポートされている表に LOB データが入っている場合のみ作成されます。作成されると、これらの LOB ファイルは *lobpath* ディレクトリーに入れられます。LOB ファイルには、合計 26,000 の可能な名前があります。

system.msg
ファイルまたはディレクトリー・コマンドを作成または削除するための、システム・メッセージの入ったメッセージ・ファイル。これは、アクションが EXPORT で、LOB パスが指定される場合のみ使用されます。

IMPORT 使用時に必要とされるファイル/生成されるファイル

- 入力:

db2move.lst
EXPORT アクションからの出力ファイル。

tab*nnn*.ixf
EXPORT アクションからの出力ファイル。

tab*nnnc*.*yyy*
EXPORT アクションからの出力ファイル。

- 出力:

IMPORT.out
IMPORT アクションの結果の要約。

tab*nnn*.msg
対応する表のインポート・メッセージ・ファイル。

LOAD 使用時に必要とされるファイル/生成されるファイル

- 入力:

db2move.lst
EXPORT アクションからの出力ファイル。

tab*nnn*.ixf
EXPORT アクションからの出力ファイル。

tab*nnnc*.*yyy*
EXPORT アクションからの出力ファイル。

- 出力:

LOAD.out
LOAD アクションの結果の要約。

tab*nnn*.msg
対応する表の **LOAD** メッセージ・ファイル。

COPY 使用時に必要とされるファイル/生成されるファイル

- 入力: なし
- 出力:

COPYSCHEMA.msg
COPY 操作中に生成されたメッセージを含む出力ファイル。

COPYSCHEMA.err
COPY 操作中に発生した各エラーに関するエラー・メッセージが含まれる出力ファイル。これには、ターゲット・データベース上に再作成できなかった各オブジェクトに関する DDL ステートメントが含まれます。

LOADTABLE.msg
ロード・ユーティリティーのそれぞれの呼び出しによって生成されたメッセージが含まれる出力ファイル (ターゲット・データベースでのデータ再挿入に使用されます)。

LOADTABLE.err
ロード中に失敗した表の名前、またはターゲット・データベースにまだデータ挿入する必要がある表の名前が含まれる出力ファイル。詳しくは『スキーマのコピー操作が失敗した場合の再開方法』のトピックを参照してください。

これらのファイルは、タイム・スタンプされ、1 つの実行から生成されたすべてのファイルには同一のタイム・スタンプが付きます。

自動生成スクリプトを使用したリダイレクト・リストアの実行

リダイレクト・リストア操作を実行するときは、バックアップ・イメージに保管されている物理コンテナーの場所を指定し、変更される各表スペースのすべてのコンテナーを提供する必要があります。

始める前に

リダイレクト・リストアを実行できるのは、事前に DB2 バックアップ・ユーティリティーを使って、データベースのバックアップをとってある場合に限ります。

このタスクについて

- データベースが存在する場合、スクリプトを生成するには、データベースに接続できなければなりません。従って、データベースでアップグレードまたはクラッシュ・リカバリーが必要な場合は、リダイレクトした復元スクリプトを生成する前に、これらの操作を行う必要があります。
- パーティション・データベース環境で作業しており、ターゲット・データベースが存在しない場合、リダイレクトされた復元スクリプトをすべてのデータベース・パーティションで同時に生成するコマンドを実行することはできません。その代わり、カタログ・パーティションからはじめて、一度に 1 つのデータベース・パーティションで、リダイレクトされた復元スクリプトを生成するコマンドを実行する必要があります。

別の方法として、ターゲット・データベースと同じ名前を持つダミーのデータベースを最初に作成することもできます。ダミーのデータベースを作成した後、すべてのデータベース・パーティションに、リダイレクトされた復元スクリプトを同時に生成することができます。

- スクリプト生成のための **RESTORE DATABASE** コマンドの発行時に **REPLACE EXISTING** パラメーターを指定しても、その **REPLACE EXISTING** パラメーターはスクリプトではコメント化されます。
- セキュリティー上の理由により、パスワードは、生成されたスクリプトに現れません。パスワードは手動で入力する必要があります。
- このリストア・スクリプトには、リストアするすべての表スペースに関するストレージ・グループの関連付け情報が入っています。

手順

スクリプトを使用してリダイレクト・リストアを実行するには、以下のようにします。

1. リストア・ユーティリティーを使用してリダイレクト・リストア・スクリプトを生成する。 リストア・ユーティリティーは、コマンド行プロセッサー (CLP)、または db2Restore アプリケーション・プログラミング・インターフェース (API) を通して起動できます。 以下は、**REDIRECT** パラメーターと **GENERATE SCRIPT** パラメーターを指定した **RESTORE DATABASE** コマンドの例です。

   ```
   db2 restore db test from /home/jseifert/backups taken at 20050304090733
         redirect generate script test_node0000.clp
   ```

   これはクライアント上に test_node0000.clp というリダイレクト・リストア・スクリプトを作成します。
2. 必要な変更を行うために、テキスト・エディターでリダイレクト・リストア・スクリプトをオープンする。 変更できるのは、次のとおりです。
   - リストア・オプション
   - 自動ストレージ・パス
   - コンテナー・レイアウトおよびパス
3. 変更されたリダイレクト・リストア・スクリプトを実行します。 例えば、以下のようにします。

   ```
   db2 -tvf test_node0000.clp
   ```

RESTORE DATABASE

RESTORE DATABASE コマンドは、DB2 バックアップ・ユーティリティーを使用してバックアップされた、損傷のある、または破壊されたデータベースを再作成します。リストアされたデータベースは、バックアップ・コピーが行われた時と同じ状態になります。

このユーティリティーは、以下のサービスも実行できます。

- データベースに別のイメージを上書きしたり、バックアップ・コピーを新しいデータベースにリストアしたりします。
- DB2 バージョン 9.8 のリストア・ユーティリティーを使用して、他のいずれかの DB2 ソフトウェアのバージョンでバックアップされたバックアップ・イメージをリストアすることはできません。

- DB2 バージョン 9.5 でバックアップされたバックアップ・イメージを DB2 バージョン 9.7 でリストアします。
  - データベース・アップグレードが必要な場合、これはリストア操作の終了時に自動的に起動されます。
- バックアップ操作の時点でデータベースのロールフォワード・リカバリーが有効になっていた場合は、リストア操作が正常に完了した後に、ロールフォワード・ユーティリティーを起動することによって、データベースを元の状態に戻すことができます。
- 表スペース・レベルのバックアップをリストアします。
- **TRANSPORT** オプションを使用して、データベース・バックアップ・イメージに含まれる表スペース、ストレージ・グループ、および SQL スキーマのセットをデータベースに転送します (DB2 バージョン 9.7 フィックスパック 2 以降のフィックスパック)。 TRANSPORT オプションは DB2 pureScale環境ではサポートされません。
- このコマンドの発行時にそのデータベース名が存在している場合には、バックアップ・イメージが作成された時点のとおりに、すべてのストレージ・グループを置換して再定義します (ユーザーが別のリダイレクトを指定していない限り)。

異なるオペレーティング・システムおよびハードウェア・プラットフォームの間で DB2 データベース・システムによってサポートされるリストア操作の詳細については、 「*データ・リカバリーと高可用性 ガイドおよびリファレンス*」の『異なるオペレーティング・システムおよびハードウェア・プラットフォーム間のバックアップおよびリストア操作』を参照してください。

オペレーティング・システムまたはワード・サイズ (32 ビットか 64 ビットか) が異なる場合、増分イメージおよび「差分イメージ」(以前のキャプチャー時との差異だけをキャプチャーするイメージ) をリストアすることはできません。

ある環境から別の環境へのリストア操作を行った後は、非増分バックアップが実施されるまで、増分バックアップまたは差分バックアップを実行できません。 (同じ環境でのリストア環境の場合、この制限はありません。)

ある環境から別の環境へのリストア操作が成功した場合でも、いくつかの注意事項があります。パッケージは、使用する前に再バインドする必要があります (**BIND** コマンド、**REBIND** コマンド、または **db2rbind** ユーティリティーを使用)。 SQL プロシージャーは、ドロップしてから再作成する必要があります。また、外部ライブラリーは、新しいプラットフォーム上ですべて再ビルドする必要があります。 (同じ環境にリストアする場合、これらの点は該当しません。)

既存のデータベースと既存のコンテナーに対して実行されるリストア操作では、同じコンテナーと表スペース・マップが再利用されます。

新規のデータベースに対して実行されるリストア操作では、すべてのコンテナーが再取得され、最適化された表スペース・マップが再作成されます。既存のデータベースに対して実行されるリストア操作でも 1 つ以上のコンテナーがない場合は、すべてのコンテナーが再取得され、最適化された表スペース・マップが再作成されます。

有効範囲

このコマンドは、それが実行されたノードに対してだけ影響を与えます。

オンラインで SYSCATSPACE をリストアすることはできません。

許可

既存のデータベースにリストアするには、次の権限のいずれかが必要です。

- SYSADM
- SYSCTRL
- SYSMAINT

新規のデータベースにリストアするには、次の権限のいずれかが必要です。

- SYSADM
- SYSCTRL

ユーザー名を指定する場合、そのユーザーにはデータベースに対する CONNECT 権限が必要です。

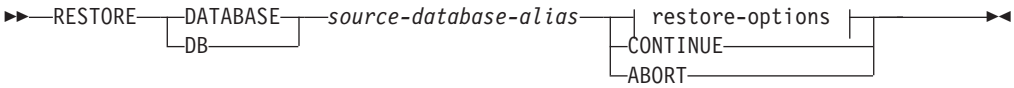
必要な接続

必要な接続は、リストア・アクションの種類によって異なります。

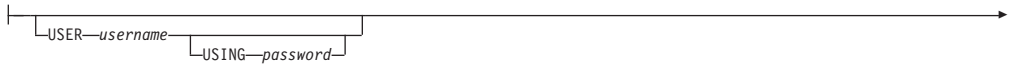
- 既存のデータベースにリストアするには、データベース接続が必要です。このコマンドは、指定されたデータベースへの排他接続を自動的に確立します。
- 新しいデータベースにリストアするには、インスタンスおよびデータベース接続が必要です。データベースを作成するには、インスタンス接続が必要です。

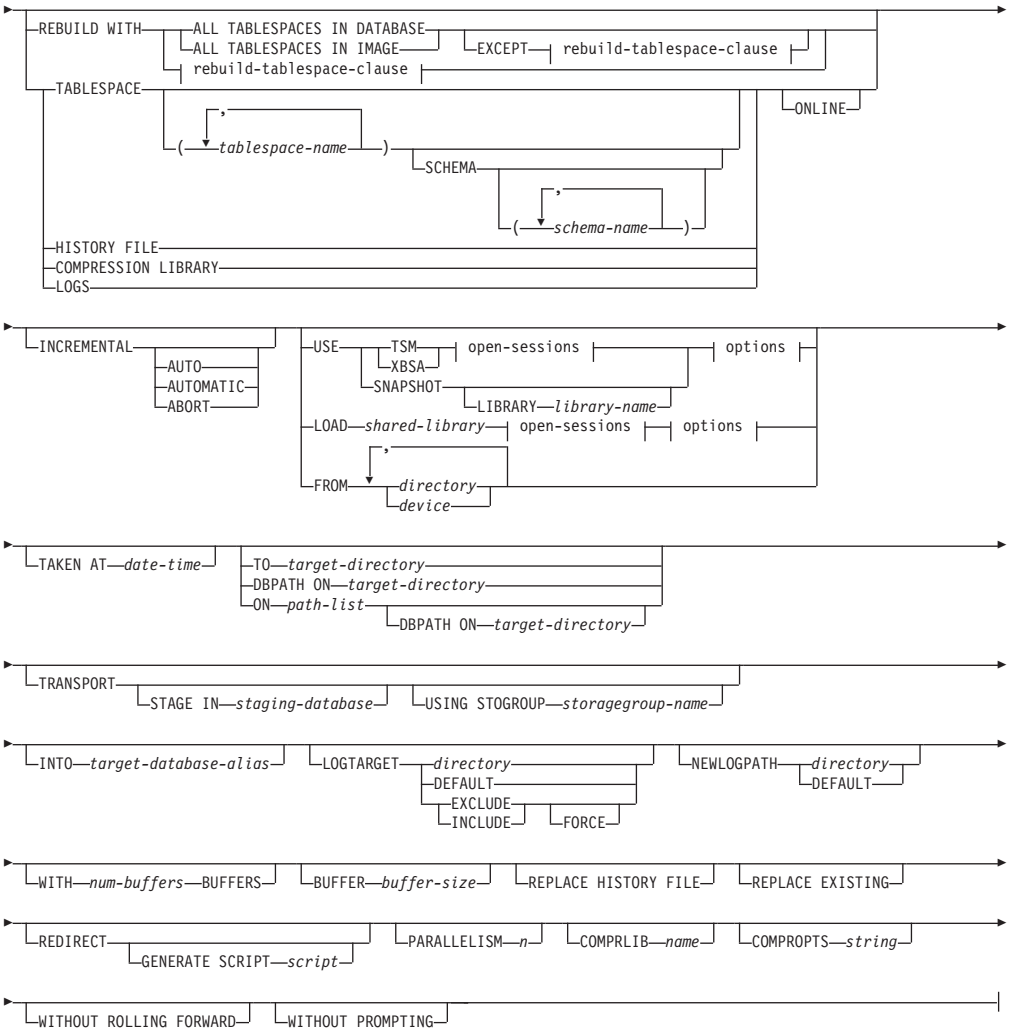
  現行のインスタンスとは異なるインスタンスで新規のデータベースへのリストアを行うには、まず、新規のデータベースを存在させるインスタンスにアタッチすることが必要です。新規インスタンスは、ローカルでもリモートでもかまいません。現行のインスタンスは、**DB2INSTANCE** 環境変数の値によって定義されます。
- スナップショット・リストアの場合、インスタンス とデータベース の接続が必要です。

コマンド構文

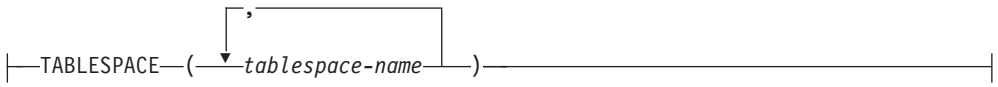


restore-options:





rebuild-tablespace-clause:



open-sessions:

options:

コマンド・パラメーター

DATABASE *source-database-alias*
バックアップが取得されるソース・データベースの別名です。

CONTINUE
コンテナーが再定義されていること、およびリダイレクトしたリストア操作の最終ステップを実行する必要があることを指定します。

ABORT
このパラメーターは以下を指定します。

- リダイレクトしたリストア操作を停止します。これは、1 つ以上のステップを繰り返す必要があるエラーが発生したときに便利です。 **ABORT** オプションを指定して **RESTORE DATABASE** を発行した後、 **REDIRECT** オプションを指定した **RESTORE DATABASE** を含む、リダイレクトしたリストア操作の各ステップを繰り返す必要があります。
- 完了する前に増分リストア操作を終了します。

USER *username*
データベースへの接続を試みるときに使用するユーザー名を指定します。

USING *password*
ユーザー名を認証するために使用するパスワード。パスワードを省略すると、ユーザーに入力を求めるプロンプトが出ます。

REBUILD WITH ALL TABLESPACES IN DATABASE
イメージをリストアする時点でデータベースが認識しているすべての表スペースを使って、データベースをリストアします。データベースが既に存在する場合、このリストアによってそれが上書きされます。

REBUILD WITH ALL TABLESPACES IN DATABASE EXCEPT
rebuild-tablespace-clause
イメージをリストアする時点でデータベースが認識しているすべての表スペースを使って、データベースをリストアします。ただし、リストで指定されているものは除外されます。データベースが既に存在する場合、このリストアによってそれが上書きされます。

REBUILD WITH ALL TABLESPACES IN IMAGE
リストアされるイメージに含まれる表スペースだけを使ってデータベースをリストアします。データベースが既に存在する場合、このリストアによってそれが上書きされます。

REBUILD WITH ALL TABLESPACES IN IMAGE EXCEPT
rebuild-tablespace-clause
リストアされるイメージに含まれる表スペースだけを使ってデータベースをリストアします。ただし、リストで指定されているものは除外されます。データベースが既に存在する場合、このリストアによってそれが上書きされます。

REBUILD WITH *rebuild-tablespace-clause*
指定された表スペースのリストだけを使ってデータベースをリストアします。データベースが既に存在する場合、このリストアによってそれが上書きされます。

TABLESPACE *tablespace-name*
リストアされる表スペースを指定するときに使用する名前のリストです。

TRANSPORT オプションを指定する場合、表スペース名は必須です。

SCHEMA *schema-name*
リストアされるスキーマを指定するために使用される名前のリスト。

TRANSPORT オプションを指定する場合、スキーマ名は必須です。 **SCHEMA** オプションは、**TRANSPORT** オプションが指定された場合のみ有効です。

ONLINE
このキーワードは、表スペース・レベルのリストア操作を行う場合のみ適用でき、これを指定するとオンラインでバックアップ・イメージがリストアできます。これは、他のエージェントが、バックアップ・イメージのリストア中にデータベースに接続できることや、指定された表スペースのリストア中に他の表スペースのデータを使用できることを意味します。

HISTORY FILE
このキーワードは、バックアップ・イメージから履歴ファイルのみをリストアする場合に指定します。

COMPRESSION LIBRARY
このキーワードは、バックアップ・イメージから圧縮ライブラリーだけをリストアする場合に指定します。バックアップ・イメージの中にオブジェクトが存在している場合、それはデータベース・ディレクトリーの中にリストアされます。バックアップ・イメージの中にオブジェクトが存在しない場合、リストア操作は失敗します。

LOGS このキーワードは、バックアップ・イメージに含まれている一連のログ・ファイルだけをリストアする場合に指定します。バックアップ・イメージの中にログ・ファイルが含まれていない場合、リストア操作は失敗します。このオプションを指定する場合は、**LOGTARGET** オプションも指定する必要があります。

INCREMENTAL
INCREMENTAL は、追加のパラメーターを使用しないで手動累積リストア操作を指定します。手動リストアの際、ユーザーはリストアに含まれるイメージごとに各リストア・コマンドを手動で発行する必要があります。以下の順序でこれを行ってください。最後、1 番目、2 番目、以下同様に最後のイメージまで。

INCREMENTAL AUTOMATIC/AUTO
自動累積リストア操作を指定します。

INCREMENTAL ABORT
手動累積リストア操作を指定します。

USE

TSM ターゲット・デバイスとして Tivoli Storage Manager (TSM) を使用して、データベースがリストアされるように指定します。

XBSA XBSA インターフェースを使用するように指定します。バックアップ・サービス API (XBSA) は、バックアップやアーカイブの目的で

データ・ストレージ管理を必要とするアプリケーションまたは機能のための、オープン・アプリケーション・プログラミング・インターフェースです。

SNAPSHOT
データがスナップショット・バックアップからリストアされるように指定します。

SNAPSHOT パラメーターを以下のいずれかのパラメーターと一緒に使用することはできません。

- **INCREMENTAL**
- **TO**
- **ON**
- **DBPATH ON**
- **INTO**
- **NEWLOGPATH**
- **WITH** *num-buffers* **BUFFERS**
- **BUFFER**
- **REDIRECT**
- **REPLACE HISTORY FILE**
- **COMPRESSION LIBRARY**
- **PARALLELISM**
- **COMPRLIB**
- **OPEN** *num-sessions* **SESSIONS**
- **HISTORY FILE**
- **LOGS**

また、**SNAPSHOT** パラメーターは、表スペース・リストが関係するリストア操作で使用することはできません。これには、**REBUILD WITH** オプションが含まれます。

スナップショット・バックアップ・イメージからデータをリストアするときのデフォルト動作は、すべてのコンテナー、ローカル・ボリューム・ディレクトリー、およびデータベース・パス (DBPATH) を含む、データベースを構成するすべてのパスの完全データベース・オフライン・リストアです。**LOGTARGET INCLUDE** パラメーターを指定した場合を除き、ログはスナップショット・リストアから除外されます。**LOGTARGET EXCLUDE** パラメーターが、すべてのスナップショット・リストアのデフォルトです。タイム・スタンプを指定すると、そのタイム・スタンプの付いたスナップショット・バックアップ・イメージがリストアに使用されます。

LIBRARY *library-name*
IBM Data Server には、以下のストレージ・ハードウェアのための DB2 ACS API ドライバーが組み込まれています。

- IBM TotalStorage SAN ボリューム・コントローラー
- IBM Enterprise Storage Server® Model 800

- IBM Storwize® V7000
- IBM System Storage® DS6000™
- IBM System Storage DS8000®
- IBM System Storage N Series
- IBM XIV®

他のストレージ・ハードウェアを使用していて、そのストレージ・ハードウェア用の DB2 ACS API ドライバーがある場合、**LIBRARY** パラメーターを使用してその DB2 ACS API ドライバーを指定できます。

LIBRARY パラメーターの値は、完全修飾ライブラリー・ファイル名です。

OPTIONS

"*options-string*"
リストア操作で使用するオプションを指定します。ストリングは、二重引用符なしで、入力されたとおりに渡されます。

@*file-name*
リストア操作で使用するオプションが、 DB2 サーバー上のファイルに含まれていることを指定します。このストリングは、ベンダー・サポートのライブラリーに渡されます。ファイル名は完全修飾ファイル名でなければなりません。

VENDOROPT データベース構成パラメーターを使用してスナップショット・リストア操作でのベンダー固有のオプションを指定することはできません。代わりに、リストア・ユーティリティーの **OPTIONS** パラメーターを使用する必要があります。

OPEN *num-sessions* **SESSIONS**
TSM またはベンダー製品とともに使用する入出力セッションの数を指定します。

FROM *directory/device*
バックアップ・イメージがあるディレクトリーまたは装置の完全修飾パス名。 **USE TSM**、**FROM**、および **LOAD** を省略した場合のデフォルト値は、クライアント・マシンの現行作業ディレクトリーです。このターゲット・ディレクトリーまたは装置は、ターゲット・サーバー/インスタンス上に存在している必要があります。

複数の項目が指定され、項目の最後がテープ装置である場合には、他のテープが要求されます。有効な応答オプションは、次のとおりです。

c 続行。警告メッセージを生成した装置を使用し続けます (例えば、新しいテープがマウントされた場合)。

d 装置の終了。警告メッセージの原因となった装置の使用だけ を停止します (例えば、もうテープがない場合に停止する、など)。

t 終了。ユーザーが、ユーティリティーによって要求された何らかのアクションを実行しなかった場合、リストア操作を異常終了します。

LOAD *shared-library*

使用するバックアップおよびリストア I/O ベンダー関数を含む共有ライブラリー (Windows オペレーティング・システムでは DLL) の名前。名前には絶対パスを含めることができます。絶対パスを指定しない場合、ユーザー出口プログラムが置かれているパスがデフォルト値として使われます。

TAKEN AT *date-time*

データベース・バックアップ・イメージのタイム・スタンプです。タイム・スタンプはバックアップ操作が正常に終了した後に表示され、バックアップ・イメージのパス名の一部になっています。 *yyyymmddhhmmss* の形式で指定されます。タイム・スタンプを部分的に指定することもできます。例えば、2 つの異なるタイム・スタンプ 20021001010101 および 20021002010101 で指定されるバックアップ・イメージが存在する場合、`20021002` を指定することで、タイム・スタンプ `20021002010101` のイメージが使用できます。このパラメーターに値を指定しない場合は、ソース・メディア上のバックアップ・イメージは 1 つだけでなければなりません。

TO *target-directory*

このパラメーターは、ターゲット・データベース・ディレクトリーを指定します。ユーティリティーが存在するデータベースへリストアしている場合には、このパラメーターは無視されます。指定するドライブおよびディレクトリーは、ローカルのものでなければなりません。自動ストレージが有効になったデータベースがバックアップ・イメージに含まれる場合、データベース・ディレクトリーだけが変更され、そのデータベースに関連したストレージ・パスは変更されません。

DBPATH ON *target-directory*

このパラメーターは、ターゲット・データベース・ディレクトリーを指定します。ユーティリティーが存在するデータベースへリストアしている場合には、このパラメーターは無視されます。指定するドライブおよびディレクトリーは、ローカルのものでなければなりません。自動ストレージが有効になったデータベースがバックアップ・イメージに含まれ、**ON** パラメーターが指定されない場合、このパラメーターは **TO** パラメーターと同じ意味になり、データベース・ディレクトリーだけが変更されます。そのデータベースに関連したストレージ・パスは変更されません。

ON *path-list*

このパラメーターは、データベースに関連付けられているストレージ・パスを再定義します。データベースに複数のストレージ・グループが含まれている場合、このオプションは、定義された各ストレージ・グループがその新しいストレージ・グループ・パスとして *path-list* を使用するように、すべてのストレージ・グループを指定のパスにリダイレクトします。ストレージ・グループが定義されていないデータベースや、自動ストレージが有効になっていないデータベースに対してこのパラメーターを使用した場合、エラー (SQL20321N) が発生します。バックアップ・イメージ内に定義された既存のストレージ・パスはもはや使用されなくなり、自動ストレージ表スペースは新しいパスに自動的にリダイレクトされます。自動ストレージ・データベースに対してこのパラメーターを指定しない場合、ストレージ・パスはバックアップ・イメージ内に定義されたままの状態になります。 1 つのパス、

またはコンマで区切った複数のパスを指定できます。それぞれのパスは絶対パス名でなければならず、ローカルに存在しなければなりません。

このオプションを **REDIRECT** オプションと一緒に指定した場合、最初の **RESTORE ... REDIRECT** コマンドが呼び出し元に戻るまでこのオプションは有効です。ただし、SET STOGROUP PATHS ステートメントまたは SET TABLESPACE CONTAINERS ステートメントが発行されると無効になります。その後ストレージ・グループ・パスがリダイレクトされた場合には、そうした変更によって、最初の **RESTORE ... ON** *path-list* コマンドで指定されたあらゆるパスはオーバーライドされます。

リストア操作中に再定義されたパスを持つストレージ・グループについては、それ以降のロールフォワード操作においてストレージ・パス関連の操作が再生されることはありません。

データベースがディスクにまだ存在せず、**DBPATH ON** パラメーターが指定されていない場合には、最初のパスがターゲット・データベース・ディレクトリーとして使用されます。

複数パーティション・データベースの場合、「**ON path-list**」オプションを指定できるのはカタログ・パーティションについてだけです。 ON オプションを使用する場合、カタログ・パーティションは、他のどのパーティションがリストアされるよりも前にリストアする必要があります。新しいストレージ・パスでカタログ・パーティションをリストアすると、非カタログ・データベース・パーティションのすべてが RESTORE_PENDING 状態になります。その場合、非カタログ・データベース・パーティションは、**RESTORE** コマンドに **ON** 文節を指定することなく並列してリストアできます。

一般的に、複数パーティション・データベースでは、どのパーティションにも同じストレージ・パスを使用する必要があり、それらはすべて、**RESTORE DATABASE** コマンドの実行前に存在している必要があります。その例外の 1 つとして、ストレージ・パス内でデータベース・パーティション式を使用する場合があります。その使用によって、処理結果のパス名が各パーティションごとに異なるように、データベース・パーティション番号をストレージ・パスにおいて反映することができます。

RESTORE コマンドに ON 節を指定して使用すると、リダイレクト・リストア操作と同じ意味になります。

ON パラメーターを使用してスキーマ転送のストレージ・パスを再定義することはできません。スキーマ転送は、ターゲット・データベースの既存のストレージ・パスを使用します。

INTO *target-database-alias*
ターゲット・データベースの別名です。ターゲット・データベースが存在しない場合には、作成されます。

データベース・バックアップを既存のデータベースにリストアするとき、リストアされたデータベースは既存のデータベースの別名およびデータベース名を継承します。データベース・バックアップを存在していないデータベースにリストアするとき、新規のデータベースが指定した別名およびデータベース名を使用して作成されます。新しいデータベース名は、リストア先のシステムで固有のものでなければなりません。

TRANSPORT INTO *target-database-alias*

転送操作で使用する既存のターゲット・データベース別名を指定します。転送される表スペースとスキーマがデータベースに追加されます。

TABLESPACE および **SCHEMA** オプションが、有効な転送可能セットを表す表スペース名とスキーマ名を指定していなければなりません。そうでない場合は、転送操作は失敗します (SQLCODE=SQL2590N rc=1)。

システム・カタログは転送できません。(SQLCODE=SQL2590N rc=4。)

RESTORE コマンドによってスキーマの妥当性検査が行われた後、転送される表スペース内のオブジェクトを表すシステム・カタログ項目が、ターゲット・データベースに作成されます。スキーマ再作成の完了後、ターゲット・データベースは物理表スペース・コンテナーの所有権を得ます。

リストアされる表スペースに含まれる物理オブジェクトと論理オブジェクトがターゲット・データベースに再作成され、表スペースの定義とコンテナーがターゲット・データベースに追加されます。オブジェクト作成時に障害が発生した場合、または DDL の再生が発生した場合は、エラーが返されます。

STAGE IN *staging-database*

転送操作のソースであるバックアップ・イメージのための一時ステージング・データベースの名前を指定します。 **STAGE IN** オプションを指定した場合、転送操作が完了しても一時データベースはドロップされません。このデータベースは、転送が完了したら必要ではなくなるので、DBA がドロップできます。

STAGE IN オプションを指定しない場合は、以下のようになります。

- データベース名は、SYSTG*xxx* の形式になります (*xxx* は整数値)。
- 転送操作完了後、一時ステージング・データベースはドロップされます。

USING STOGROUP *storagegroup-name*

自動ストレージ表スペースの場合、すべての表スペースに関連付けられるターゲット・ストレージ・グループが転送されることを指定します。ストレージ・グループが指定されない場合には、ターゲット・データベースの、現在指定されているデフォルトのストレージ・グループが使用されます。この節は自動ストレージ表スペースのみに適用され、スキーマの `transport` 操作中のみ有効です。

表スペース・データが格納されるストレージ・グループを指定します。*storagegroup-name* には、**TRANSPORT** 操作の *target-database-alias* に存在するストレージ・グループを指定する必要があります (SQLSTATE 42704)。これは、1 部構成の名前です。

LOGTARGET *directory*

スナップショット以外のリストアの場合:

バックアップ・イメージからログ・ファイルを抽出する際のターゲット・ディレクトリーとして使用する、データベース・サーバー上の既存のディレクトリーの絶対パス名。このオプションを指定する場合、バックアップ・イメージ内のログ・ファイルは、そのターゲット・ディレクトリー内に抽出されます。このオプションを指定しない場合、バックアップ・イメージ内のログ・ファイルは抽出されません。バックアップ・イメージからログ・ファイ

ルだけを抽出する場合は、 **LOGS** オプションを指定してください。このオプションは、データベース・パーティション番号とログ・ストリーム ID を自動的にパスに付加します。

DEFAULT
ログ・ファイルをバックアップ・イメージからデータベースのデフォルトのログ・ディレクトリー (/home/db2user/db2inst/NODE0000/SQL00001/LOGSTREAM0000 など) にリストアします。

スナップショット・リストアの場合:

INCLUDE
スナップショット・イメージからログ・ディレクトリー・ボリュームをリストアします。このオプションが指定されていて、バックアップ・イメージにログ・ディレクトリーが含まれている場合、それらはリストアされます。ディスク上に既存のログ・ディレクトリーとログ・ファイルは、バックアップ・イメージ中のログ・ディレクトリーと競合するのでなければ、変更なしでそのままになります。ディスク上に既存のログ・ディレクトリーがバックアップ・イメージ中のログ・ディレクトリーと競合する場合は、エラーが戻されます。

EXCLUDE
ログ・ディレクトリー・ボリュームをリストアしません。このオプションを指定すると、バックアップ・イメージからログ・ディレクトリーはリストアされません。ディスク上に既存のログ・ディレクトリーとログ・ファイルは、バックアップ・イメージ中のログ・ディレクトリーと競合するのでなければ、変更なしでそのままになります。データベースに属する 1 つのパスがリストアされ、そのために暗黙のうちに 1 つのログ・ディレクトリーがリストアされ、その結果、ログ・ディレクトリーが上書きされることになる場合、エラーが戻されます。

FORCE
スナップショット・イメージをリストアする時に現行データベースの既存のログ・ディレクトリーを上書きおよび置換することを許可します。このオプションを使用しなければ、スナップショット・イメージのログ・ディレクトリーと矛盾するディスク上の既存のログ・ディレクトリーおよびログ・ファイルが原因で、リストアは失敗します。このオプションを使用して、リストアでこれらの既存のログ・ディレクトリーを上書きおよび置換できるように指示します。

注: このオプションは注意して使用し、リカバリーに必要になる可能性があるすべてのログを常にバックアップおよびアーカイブしてください。

スナップショット・リストアでは、ディレクトリー・オプションのデフォルト値は **LOGTARGET EXCLUDE** です。

NEWLOGPATH *directory*
リストア操作後にアクティブ・ログ・ファイルに使用されるディレクトリー

の絶対パス名。このパラメーターの機能は **newlogpath** データベース構成パラメーターと同じです。このパラメーターは、バックアップ・イメージのログ・パスが、リストア後の使用に適していない場合に使用することができます。例えば、パスがもはや有効でない、または別のデータベースによって使用されている、という場合などです。

注: **newlogpath** コマンド・パラメーターが設定されると、**newlogpath** データベース構成パラメーターを更新する場合と同様に、**logpath** の値にはノード番号が自動的には追加されなくなります。詳細については、『newlogpath - データベース・ログ・パスの変更』を参照してください。

DEFAULT
リストアが完了した後、データベースはデフォルトのログ・ディレクトリー `/home/db2user/db2inst/NODE0000/SQL00001/LOGSTREAM0000` をロギングに使用します。

WITH *num-buffers* **BUFFERS**
使用するバッファーの数です。値を明示的に指定しない場合、DB2 データベース・システムはこのパラメーターの最適値を自動的に選択します。複数のソースが読み取られる場合や、**PARALLELISM** の値が増やされている場合は、パフォーマンスを向上させるために複数のバッファーを使用することができます。

BUFFER *buffer-size*
リストア操作に使用するバッファーのサイズ (ページ数)。値を明示的に指定しない場合、DB2 データベース・システムはこのパラメーターの最適値を自動的に選択します。このパラメーターの最小値は 8 ページです。

リストア・バッファー・サイズは、バックアップ操作中に指定したバックアップ・バッファー・サイズに正の整数を乗算したサイズでなければなりません。誤ったバッファー・サイズを指定すると、許容可能な最小のサイズで割り振られます。

REPLACE HISTORY FILE
リストア操作において、ディスク上の履歴ファイルを、バックアップ・イメージの履歴ファイルで置換することを指定します。

REPLACE EXISTING
ターゲット・データベースの別名と同じ別名を持つデータベースが既に存在している場合、このパラメーターは、リストア・ユーティリティーが既存のデータベースをリストアしたデータベースに置換するように指定します。これはリストア・ユーティリティーを起動するスクリプトで便利です。コマンド行プロセッサーは、ユーザーに既存のデータベースの削除を検証するよう求めるプロンプトを出さないためです。 **WITHOUT PROMPTING** パラメーターが指定された場合、**REPLACE EXISTING** を指定する必要はありませんが、その場合、ユーザー介入を標準的に必要とするイベントが起こるとこの操作は失敗します。

REDIRECT
リダイレクトしたリストア操作を指定します。リダイレクトしたリストア操作を完了するには、このコマンドの後に 1 つ以上の **SET TABLESPACE**

CONTAINERS コマンドまたは **SET STOGROUP PATHS** コマンドを続け、次に **CONTINUE** オプションを指定して **RESTORE DATABASE** コマンドを続ける必要があります。以下に例を示します。

```
RESTORE DB SAMPLE REDIRECT

SET STOGROUP PATHS FOR sg_hot ON '/ssd/fs1', '/ssd/fs2'
SET STOGROUP PATHS FOR sg_cold ON '/hdd/path1', '/hdd/path2'

RESTORE DB SAMPLE CONTINUE
```

バックアップ・イメージの作成以降にストレージ・グループが名前変更された場合には、**SET STOGROUP PATHS** コマンドに指定されるストレージ・グループ名は、最新の名前ではなくバックアップ・イメージからのストレージ・グループ名を参照します。

同一のリダイレクトしたリストア操作に関連したコマンドはすべて、同じウィンドウまたは CLP セッションから起動しなければなりません。

GENERATE SCRIPT *script*
指定されたファイル名を使用して、リダイレクト・リストア・スクリプトを作成します。スクリプト名は相対パスまたは絶対パスであり、そのスクリプトはクライアント・サイドで生成されます。クライアント・サイドでそのファイルを作成できない場合には、エラー・メッセージ (SQL9304N) が戻されます。ファイルが既に存在する場合は上書きされます。使用法について詳しくは、下記の例を参照してください。

WITHOUT ROLLING FORWARD
データベースを、正常にリストアされた後ロールフォワード・ペンディング状態にしないように指定します。

正常なリストアに続いて、データベースがロールフォワード・ペンディング状態にある場合には、データベースが使用できるようになる前に、**ROLLFORWARD** コマンドを起動する必要があります。

オンライン・バックアップ・イメージからのリストアでこのオプションを指定した場合、エラー SQL2537N が戻されます。

注: リカバリー可能データベースのバックアップ・イメージである場合、**REBUILD** オプションに **WITHOUT ROLLING FORWARD** を指定することはできません。

PARALLELISM *n*
リストア操作中に作成されるバッファー・マニピュレーターの数を指定します。値を明示的に指定しない場合、DB2 データベース・システムはこのパラメーターの最適値を自動的に選択します。

COMPRLIB *name*
解凍を実行するために使用するライブラリーの名前を指定します (例えば、Windows の場合は `db2compr.dll`、Linux または UNIX システムの場合は `libdb2compr.so`)。この名前は、サーバー上の 1 個のファイルを参照する完全修飾パスでなければなりません。このパラメーターを指定しない場合、DB2 データベース・システムはイメージ内に格納されているライブラリーの使用を試みます。バックアップが圧縮されていなかった場合、このパラメーターの値は無視されます。指定されたライブラリーをロードできない場合、リストア操作は失敗します。

COMPROPTS *string*

バイナリー・データのうち、解凍ライブラリーの初期設定ルーチンに渡すブロックを記述します。 DB2 データベース・システムはこのストリングをクライアントからサーバーに直接渡すため、バイト反転やコード・ページ変換の問題がある場合は、解凍ライブラリーで処理されます。データ・ブロックの最初の文字が「@」なら、データの残りの部分は、サーバー上に存在するファイルの名前を指定するものとして、DB2 データベース・システムは解釈します。その場合、DB2 データベース・システムは *string* の内容をこのファイルの内容で置き換え、新しい値を初期設定ルーチンに渡します。ストリングの最大長は 1 024 バイトです。

WITHOUT PROMPTING

リストア操作を無人で実行するように指定します。通常はユーザー介入を必要とするアクションでは、エラー・メッセージが戻されます。テープやディスケットなどの取り外し可能メディア装置を使用している場合、このオプションを指定していても、その装置が終わるとプロンプトが出されます。

例

1. 以下の例で、データベース WSDB は 0 から 3 までの番号が付けられた 4 つのデータベース・パーティションすべてに定義されています。パス /dev3/backup はすべてのデータベース・パーティションからアクセスできます。以下のオフライン・バックアップ・イメージは、/dev3/backup から入手可能です。

   ```
   wsdb.0.db2inst1.DBPART000.200802241234.001
   wsdb.0.db2inst1.DBPART001.200802241234.001
   wsdb.0.db2inst1.DBPART002.200802241234.001
   wsdb.0.db2inst1.DBPART003.200802241234.001
   ```

   最初にカタログ・パーティションをリストアしてから WSDB データベースの他のすべてのデータベース・パーティションを /dev3/backup ディレクトリーからリストアするには、いずれかのデータベース・パーティションから以下のコマンドを実行します。

   ```
   db2_all '<<+0< db2 RESTORE DATABASE wsdb FROM /dev3/backup
   TAKEN AT 20020331234149
     INTO wsdb REPLACE EXISTING'
   db2_all '<<+1< db2 RESTORE DATABASE wsdb FROM /dev3/backup
   TAKEN AT 20020331234427
     INTO wsdb REPLACE EXISTING'
   db2_all '<<+2< db2 RESTORE DATABASE wsdb FROM /dev3/backup
   TAKEN AT 20020331234828
     INTO wsdb REPLACE EXISTING'
   db2_all '<<+3< db2 RESTORE DATABASE wsdb FROM /dev3/backup
   TAKEN AT 20020331235235
     INTO wsdb REPLACE EXISTING'
   ```

   db2_all ユーティリティーは、指定された各データベース・パーティションへのリストア・コマンドを出します。 db2_all を使用してリストアを実行する場合は、常に **REPLACE EXISTING** や **WITHOUT PROMPTING** を指定してください。これを指定しないと、プロンプトが表示された場合に操作がハングしたように見えます。それは、**db2_all** でユーザー・プロンプトがサポートされていないためです。

2. 以下は、別名が MYDB であるデータベースの典型的なリダイレクト・リストアのシナリオです。

a. 次のように、**REDIRECT** オプションを指定して **RESTORE DATABASE** コマンドを発行する。

```
restore db mydb replace existing redirect
```

ステップ 1 が正常終了した後でステップ 3 が完了する前に、次を発行してリストア操作を打ち切ることができる。

```
restore db mydb abort
```

b. 再定義する必要があるコンテナーを持つ表スペースごとに、 **SET TABLESPACE CONTAINERS** コマンドを発行する。以下に例を示します。

```
set tablespace containers for 5 using
   (file 'f:¥ts3con1' 20000, file 'f:¥ts3con2' 20000)
```

リストアしたデータベースのコンテナーが、このステップで指定したものであることを検査するために、 **LIST TABLESPACE CONTAINERS** コマンドを発行する。

c. ステップ 1 および 2 が正常終了した後、次を発行します。

```
restore db mydb continue
```

これはリダイレクト・リストア操作の最終ステップです。

d. ステップ 3 が失敗した場合、またはリストア操作を打ち切った場合、リダイレクト・リストアはステップ 1 から再始動できます。

3. 以下の例は、リカバリー可能データベース用の週次の増分バックアップ・ストラテジーのサンプルです。週 1 回の全データベース・バックアップ操作、1 日 1 回の非累積 (差分) バックアップ操作、および週 2 回の累積 (増分) バックアップ操作が含まれています。

```
(Sun) backup db mydb use tsm
(Mon) backup db mydb online incremental delta use tsm
(Tue) backup db mydb online incremental delta use tsm
(Wed) backup db mydb online incremental use tsm
(Thu) backup db mydb online incremental delta use tsm
(Fri) backup db mydb online incremental delta use tsm
(Sat) backup db mydb online incremental use tsm
```

金曜日の午前中に作成されたイメージを自動データベース・リストアするには、次のようにします。

```
restore db mydb incremental automatic taken at (Fri)
```

金曜日の午前中に作成されたイメージを手動データベース・リストアするには、次のようにします。

```
restore db mydb incremental taken at (Fri)
restore db mydb incremental taken at (Sun)
restore db mydb incremental taken at (Wed)
restore db mydb incremental taken at (Thu)
restore db mydb incremental taken at (Fri)
```

4. リモート・サイトに移動することを意図したバックアップ・イメージを作成し、それにログを含めるには、次のようにします。

```
backup db sample online to /dev3/backup include logs
```

このバックアップ・イメージをリストアするには、**LOGTARGET** パスを指定し、**ROLLFORWARD** でそのパスを指定します。

```
restore db sample from /dev3/backup logtarget /dev3/logs
rollforward db sample to end of logs and stop overflow log path /dev3/logs
```

5. ログを含むバックアップ・イメージから、ログ・ファイルだけを取り出すには、

   ```
   restore db sample logs from /dev3/backup logtarget /dev3/logs
   ```

6. 次の例では、データベース SAMPLE のバックアップ操作のために、同一のターゲット・ディレクトリーを 3 回指定しています。データは 3 つのターゲット・ディレクトリーに並行してバックアップされ、それら 3 つのバックアップ・イメージは拡張子 .001、 .002、および .003 が付けられて生成されます。

   ```
   backup db sample to /dev3/backup, /dev3/backup, /dev3/backup
   ```

   ターゲット・ディレクトリーからバックアップ・イメージをリストアするには、次を発行します。

   ```
   restore db sample from /dev3/backup, /dev3/backup, /dev3/backup
   ```

7. リストア操作で使用する TSM 情報を指定するには、 **USE TSM OPTIONS** キーワードを使用します。 Windows プラットフォームでは、-fromowner オプションを指定しないでください。

   - 区切り文字付きストリングを指定する場合、

     ```
     restore db sample use TSM options '"-fromnode=bar -fromowner=dmcinnis"'
     ```

   - 完全修飾ファイル名を指定する場合、

     ```
     restore db sample use TSM options @/u/dmcinnis/myoptions.txt
     ```

     ファイル myoptions.txt には、 -fromnode=bar -fromowner=dmcinnis というストリングが含まれています。

8. 以下に示すのは、新しいストレージ・パスによる、複数パーティション自動ストレージ対応データベースの簡単なリストアです。もともとこのデータベースは、1 つのストレージ・パス /myPath0: を使用して作成されたものです。

   - カタログ・パーティションで、次のコマンドを発行します。restore db mydb on /myPath1,/myPath2
   - カタログでないすべてのパーティションで、次のコマンドを発行します。

     ```
     restore db mydb
     ```

9. 非自動ストレージ・データベースにおいて以下のコマンドを発行すると、そのスクリプト出力は、

   ```
   restore db sample from /home/jseifert/backups taken at 20050301100417 redirect
   generate script SAMPLE_NODE0000.clp
   ```

   下記のようなものになります。

   ```
   -- ****************************************************************************
   -- ** automatically created redirect restore script
   -- ****************************************************************************
   UPDATE COMMAND OPTIONS USING S ON Z ON SAMPLE_NODE0000.out V ON;
   SET CLIENT ATTACH_DBPARTITIONNUM  0;
   SET CLIENT CONNECT_DBPARTITIONNUM 0;
   -- ****************************************************************************
   -- ** initialize redirected restore
   -- ****************************************************************************
   RESTORE DATABASE SAMPLE
   -- USER  '<username>'
   -- USING '<password>'
   FROM '/home/jseifert/backups'
   ```

```
TAKEN AT 20050301100417
-- DBPATH ON '<target-directory>'
INTO SAMPLE
-- NEWLOGPATH '/home/jseifert/jseifert/SAMPLE/NODE0000/LOGSTREAM0000/'
-- WITH <num-buff> BUFFERS
-- BUFFER <buffer-size>
-- REPLACE HISTORY FILE
-- REPLACE EXISTING
REDIRECT
-- PARALLELISM <n>
-- WITHOUT ROLLING FORWARD
-- WITHOUT PROMPTING
;
-- ******************************************************************************
-- ** tablespace definition
-- ******************************************************************************
-- ******************************************************************************
-- ** Tablespace name                         = SYSCATSPACE
-- **   Tablespace ID                         = 0
-- **   Tablespace Type                       = System managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
-- **   Total number of pages                 = 5572
-- ******************************************************************************
SET TABLESPACE CONTAINERS FOR 0
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0000.0'
);
-- ******************************************************************************
-- ** Tablespace name                         = TEMPSPACE1
-- **   Tablespace ID                         = 1
-- **   Tablespace Type                       = System managed space
-- **   Tablespace Content Type               = System Temporary data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
-- **   Total number of pages                 = 0
-- ******************************************************************************
SET TABLESPACE CONTAINERS FOR 1
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0001.0'
);
-- ******************************************************************************
-- ** Tablespace name                         = USERSPACE1
-- **   Tablespace ID                         = 2
-- **   Tablespace Type                       = System managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
-- **   Total number of pages                 = 1
-- ******************************************************************************
SET TABLESPACE CONTAINERS FOR 2
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0002.0'
);
-- ******************************************************************************
-- ** Tablespace name                         = DMS
-- **   Tablespace ID                         = 3
-- **   Tablespace Type                       = Database managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
```

```
-- **   Tablespace Extent size (pages)           = 32
-- **   Using automatic storage                  = No
-- **   Auto-resize enabled                      = No
-- **   Total number of pages                    = 2000
-- **   Number of usable pages                   = 1960
-- **   High water mark (pages)                  = 96
-- ******************************************************************************
SET TABLESPACE CONTAINERS FOR 3
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  FILE   /tmp/dms1                                                    1000
, FILE   /tmp/dms2                                                    1000
);
-- ******************************************************************************
-- ** Tablespace name                          = RAW
-- **   Tablespace ID                            = 4
-- **   Tablespace Type                          = Database managed space
-- **   Tablespace Content Type                  = Any data
-- **   Tablespace Page size (bytes)             = 4096
-- **   Tablespace Extent size (pages)           = 32
-- **   Using automatic storage                  = No
-- **   Auto-resize enabled                      = No
-- **   Total number of pages                    = 2000
-- **   Number of usable pages                   = 1960
-- **   High water mark (pages)                  = 96
-- ******************************************************************************
SET TABLESPACE CONTAINERS FOR 4
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  DEVICE '/dev/hdb1'                                          1000
, DEVICE '/dev/hdb2'                                          1000
);
-- ******************************************************************************
-- ** start redirect restore
-- ******************************************************************************
RESTORE DATABASE SAMPLE CONTINUE;
-- ******************************************************************************
-- ** end of file
-- ******************************************************************************
```

10. 自動ストレージ・データベースにおいて以下のコマンドを発行すると、そのスクリプト出力は、

```
restore db test from /home/jseifert/backups taken at 20050304090733 redirect
generate script TEST_NODE0000.clp
```

下記のようなものになります。

```
-- ******************************************************************************
-- ** automatically created redirect restore script
-- ******************************************************************************
UPDATE COMMAND OPTIONS USING S ON Z ON TEST_NODE0000.out V ON;
SET CLIENT ATTACH_MEMBER  0;
SET CLIENT CONNECT_MEMBER 0;
-- ******************************************************************************
-- ** initialize redirected restore
-- ******************************************************************************
RESTORE DATABASE TEST
-- USER  '<username>'
-- USING '<password>'
FROM '/home/jseifert/backups'
TAKEN AT 20050304090733
ON '/home/jseifert'
-- DBPATH ON <target-directory>
INTO TEST
-- NEWLOGPATH '/home/jseifert/jseifert/TEST/NODE0000/LOGSTREAM0000/'
-- WITH <num-buff> BUFFERS
-- BUFFER <buffer-size>
-- REPLACE HISTORY FILE
-- REPLACE EXISTING
```

```
REDIRECT
-- PARALLELISM <n>
-- WITHOUT ROLLING FORWARD
-- WITHOUT PROMPTING
;
-- *****************************************************************************
-- ** storage group definition
-- **   Default storage group ID                = 0
-- **   Number of storage groups                = 3
-- *****************************************************************************
-- *****************************************************************************
-- ** Storage group name                        = SG_DEFAULT
-- **   Storage group ID                        = 0
-- **   Data tag                                = None
-- *****************************************************************************
-- SET STOGROUP PATHS FOR SG_DEFAULT
-- ON '/hdd/path1'
-- ,  '/hdd/path2'
-- ;
-- *****************************************************************************
-- ** Storage group name                        = SG_HOT
-- **   Storage group ID                        = 1
-- **   Data tag                                = 1
-- *****************************************************************************
-- SET STOGROUP PATHS FOR SG_HOT
-- ON '/ssd/fs1'
-- ,  '/ssd/fs2'
-- ;
-- *****************************************************************************
-- ** Storage group name                        = SG_COLD
-- **   Storage group ID                        = 2
-- **   Data tag                                = 9
-- *****************************************************************************
-- SET STOGROUP PATHS FOR SG_COLD
-- ON '/hdd/slowpath1'
-- ;
-- *****************************************************************************
-- ** tablespace definition
-- *****************************************************************************
-- *****************************************************************************
-- ** Tablespace name                          = SYSCATSPACE
-- **   Tablespace ID                           = 0
-- **   Tablespace Type                         = Database managed space
-- **   Tablespace Content Type                 = Any data
-- **   Tablespace Page size (bytes)            = 4096
-- **   Tablespace Extent size (pages)          = 4
-- **   Using automatic storage                 = Yes
-- **   Storage group ID                        = 0
-- **   Source storage group ID                 = -1
-- **   Data tag                                = None
-- **   Auto-resize enabled                     = Yes
-- **   Total number of pages                   = 6144
-- **   Number of usable pages                  = 6140
-- **   High water mark (pages)                 = 5968
-- *****************************************************************************
-- *****************************************************************************
-- ** Tablespace name                          = TEMPSPACE1
-- **   Tablespace ID                           = 1
-- **   Tablespace Type                         = System managed space
-- **   Tablespace Content Type                 = System Temporary data
-- **   Tablespace Page size (bytes)            = 4096
-- **   Tablespace Extent size (pages)          = 32
-- **   Using automatic storage                 = Yes
-- **   Total number of pages                   = 0
-- *****************************************************************************
-- *****************************************************************************
-- ** Tablespace name                          = USERSPACE1
-- **   Tablespace ID                           = 2
-- **   Tablespace Type                         = Database managed space
-- **   Tablespace Content Type                 = Any data
-- **   Tablespace Page size (bytes)            = 4096
-- **   Tablespace Extent size (pages)          = 32
-- **   Using automatic storage                 = Yes
-- **   Storage group ID                        = 1
-- **   Source storage group ID                 = -1
```

```
-- **   Data tag                                = 1
-- **   Auto-resize enabled                     = Yes
-- **   Total number of pages                   = 256
-- **   Number of usable pages                  = 224
-- **   High water mark (pages)                 = 96
-- ******************************************************************************
-- ******************************************************************************
-- ** Tablespace name                           = DMS
-- **   Tablespace ID                           = 3
-- **   Tablespace Type                         = Database managed space
-- **   Tablespace Content Type                 = Any data
-- **   Tablespace Page size (bytes)            = 4096
-- **   Tablespace Extent size (pages)          = 32
-- **   Using automatic storage                 = No
-- **   Storage group ID                        = 2
-- **   Source storage group ID                 = -1
-- **   Data tag                                = 9
-- **   Auto-resize enabled                     = No
-- **   Total number of pages                   = 2000
-- **   Number of usable pages                  = 1960
-- **   High water mark (pages)                 = 96
-- ******************************************************************************
SET TABLESPACE CONTAINERS FOR 3
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  FILE   '/tmp/dms1'                                        1000
, FILE   '/tmp/dms2'                                        1000
);
-- ******************************************************************************
-- ** Tablespace name                           = RAW
-- **   Tablespace ID                           = 4
-- **   Tablespace Type                         = Database managed space
-- **   Tablespace Content Type                 = Any data
-- **   Tablespace Page size (bytes)            = 4096
-- **   Tablespace Extent size (pages)          = 32
-- **   Using automatic storage                 = No
-- **   Auto-resize enabled                     = No
-- **   Total number of pages                   = 2000
-- **   Number of usable pages                  = 1960
-- **   High water mark (pages)                 = 96
-- ******************************************************************************
SET TABLESPACE CONTAINERS FOR 4
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  DEVICE '/dev/hdb1'                                        1000
, DEVICE '/dev/hdb2'                                        1000
);
-- ******************************************************************************
-- ** start redirect restore
-- ******************************************************************************
RESTORE DATABASE TEST CONTINUE;
-- ******************************************************************************
-- ** end of file
-- ******************************************************************************
```

11. **SNAPSHOT** オプションを使用した **RESTORE DB** コマンドの例を以下に示します。

    ログ・ディレクトリー・ボリュームをスナップショット・イメージからリストアし、プロンプトを出しません。

    ```
    db2 restore db sample use snapshot LOGTARGET INCLUDE without prompting
    ```

    ログ・ディレクトリー・ボリュームをリストアせず、プロンプトを出しません。

    ```
    db2 restore db sample use snapshot LOGTARGET EXCLUDE without prompting
    ```

    ログ・ディレクトリー・ボリュームをリストアせず、プロンプトを出しません。 **LOGTARGET** が指定されていない場合には、デフォルトは **LOGTARGET EXCLUDE** です。

    ```
    db2 restore db sample use snapshot without prompting
    ```

矛盾するログ・ディレクトリーが含まれるスナップショット・イメージをリストアする時に、プロンプトを出さずに現行データベースの既存のログ・ディレクトリーを上書きおよび置換することを許可します。

```
db2 restore db sample use snapshot LOGTARGET EXCLUDE FORCE without prompting
```

矛盾するログ・ディレクトリーが含まれるスナップショット・イメージをリストアする時に、プロンプトを出さずに現行データベースの既存のログ・ディレクトリーを上書きおよび置換することを許可します。

```
db2 restore db sample use snapshot LOGTARGET INCLUDE FORCE without prompting
```

12. **RESTORE** コマンドに **TRANSPORT REDIRECT** オプションを指定した転送操作の例を以下に示します。

    ソース・データベース (TT_SRC) のバックアップ・イメージのストレージ・パスが /src で、ターゲット・データベース (TT_TGT) のストレージ・パスが /tgt と想定しています。

    ```
    > RESTORE DB TT_SRC TABLESPACE (AS1) SCHEMA (KRODGER) TRANSPORT INTO TT_TGT REDIRECT

    SQL1277W A redirected restore operation is being performed.  Table space
    configuration can now be viewed and table spaces that do not use automatic
    storage can have their containers reconfigured.
    DB20000I The RESTORE DATABASE command completed successfully.
    ```

    表スペース「AS1」が、/tgt/krodger/NODE0000/TT_TGT/T0000003/C0000000.LRG のようなコンテナー・パスに転送されます。

    転送する表スペースのターゲット・ストレージ・グループを指定するには、**RESTORE** コマンドの **USING STOGROUP** オプションを使用できます。以下の例では、表スペース TS1 および TS2 の両方が SG_COLD ストレージ・グループにリストアされます。

    ```
    RESTORE DB TT_SRC TABLESPACE (TS1, TS2) SCHEMA (KRODGER) TRANSPORT INTO TT_TGT USING STOGROUP SG_COLD
    ```

    注: RESTORE コマンドの **USING STOGROUP** オプションが有効であるのは、転送操作のみであるため、その他のリストア操作中にターゲット・ストレージ・グループを指定するために使用することはできません。
    ターゲット・データベースのデフォルト・ストレージ・グループへの転送を実行する場合、 **USING STOGROUP** オプションを指定する必要はありません。

    ```
    RESTORE  DB TT_SRC TABLESPACE (TS3) SCHEMA (KRODGER) TRANSPORT INTO TT_TGT
    ```

    TRANSPORT 操作中に **RESTORE** コマンドで指定するストレージ・グループ名は、ターゲット・データベースに現在定義されているものでなければなりません。バックアップ・イメージやソース・データベースに定義されていなくても構いません。

ドロップされたデータベース・パーティションを再作成する (損傷したため) 目的でパラメーター **AT DBPARTITIONNUM** を使用した場合、このデータベース・パーティションのデータベースはリストア・ペンディング状態になります。データベース・パーティションを再作成した後、このデータベース・パーティション上のデータベースをただちにリストアする必要があります。

使用上の注意

- db2 restore db *name* という形式の **RESTORE DATABASE** コマンドは、データベース・イメージを使ったフル・データベース・リストアの実行と、表スペース・イメージ内に検出される表スペースの表スペース・リストア操作を実行します。db2 restore db *name* tablespace という形式の **RESTORE DATABASE** コマンドは、イメージ内に検出される表スペースの表スペース・リストアを実行します。さらに、そのようなコマンドで表スペースのリストが指定された場合、明示的にリストされるすべての表スペースがリストアされます。
- オンライン・バックアップのリストア操作を実行した後、ロールフォワード・リカバリーを実行する必要があります。
- プロキシー・ノードをサポートする TSM 環境では、**OPTIONS** パラメーターを使用してリストア操作を使用可能にすることができます。詳しくは、『Tivoli Storage Manager クライアントの構成』のトピックを参照してください。
- バックアップ・イメージが圧縮されているなら、DB2 データベース・システムはそのことを検出し、データはリストア前に自動的に解凍されます。 db2Restore API でライブラリーが指定されている場合、データの解凍にはそれが使用されます。そうでない場合、ライブラリーがバックアップ・イメージに保管されているかどうか検査され、ライブラリーが存在する場合にはそれが使用されます。最後に、ライブラリーがバックアップ・イメージに入っていない場合には、データを解凍できず、リストア操作が失敗します。
- バックアップ・イメージから圧縮ライブラリーをリストアする場合 (**COMPRESSION LIBRARY** オプションを指定して明示的に、または圧縮バックアップの通常のリストアを実行することにより暗黙的に)、そのリストア操作は、バックアップが作成されたのと同じプラットフォームおよびオペレーティング・システム上で実行する必要があります。バックアップ作成時のプラットフォームとリストア操作実行時のプラットフォームが違っていると、それらの 2 つのシステムの間のクロスプラットフォーム・リストアが DB2 データベース・システムで通常にサポートされている場合でも、リストア操作は失敗します。
- バックアップした SMS 表スペースをリストアできるのは、SMS 表スペースに対してのみです。DMS 表スペースにはリストアできませんし、その逆も不可能です。
- ログ・ファイルを含むバックアップ・イメージからログ・ファイルをリストアする場合には、**LOGTARGET** オプションを指定する必要があります。その際、DB2 サーバー上に存在する有効な完全修飾パス名を指定する必要があります。それらの条件が満たされている場合、リストア・ユーティリティーは、イメージ内のログ・ファイルをターゲット・パスに書き込みます。ログを含まないバックアップ・イメージのリストア操作で **LOGTARGET** を指定した場合、表スペース・データのリストアが試行される前にエラーが戻されます。また、**LOGTARGET** に無効なパスや読み取り専用パスが指定された場合も、リストア操作が失敗してエラーが返されます。
- **RESTORE DATABASE** コマンド発行の時点で **LOGTARGET** パス内にログ・ファイルが存在している場合、警告プロンプトがユーザーに対して返されます。 **WITHOUT PROMPTING** が指定されている場合、この警告は戻されません。
- **LOGTARGET** を指定したリストア操作において、いずれかのログ・ファイルを抽出できない場合には、リストア操作が失敗してエラーが戻されます。バックアップ・イメージから抽出されるいずれかのログ・ファイルの名前が、**LOGTARGET** パ

ス内に存在するファイル名と同じである場合には、リストア操作が失敗してエラーが返されます。データベース・リストア・ユーティリティーは、**LOGTARGET** ディレクトリー内の既存のログ・ファイルを上書きしません。

- 保存されているログ・セットだけをバックアップ・イメージからリストアすることも可能です。ログ・ファイルだけをリストアするように指定するには、**LOGTARGET** パスに加えて **LOGS** オプションを指定します。 **LOGTARGET** パスを指定しないで **LOGS** オプションを指定すると、エラーになります。この操作モードでログ・ファイルをリストアしようとして問題が発生した場合、そのリストア操作は即座に終了し、エラーが戻されます。
- 自動増分リストア操作においては、リストア操作のターゲット・イメージに含まれるログ・ファイルだけがバックアップ・イメージから取り出されます。増分リストア処理中に参照される中間イメージに含まれるログ・ファイルは、それらの中間バックアップ・イメージから抽出されません。手動増分リストア操作の場合、**LOGTARGET** パスは、最後に発行するリストア・コマンドにのみ指定してください。
- オフラインの全データベース・バックアップおよびオフラインの増分データベース・バックアップは、より新しいデータベース・バージョンにリストアできますが、オンライン・バックアップはそれができません。複数パーティション・データベースでは、まずカタログ・パーティションを別個にリストアしてから、その後に残りのデータベース・パーティションを (並列または逐次に) リストアする必要があります。ただし、リストア操作によって実行された暗黙的なデータベース・アップグレードは、失敗する可能性があります。複数パーティション・データベースでは、1 つ以上のデータベース・パーティションでそれが失敗することがあります。この場合、**RESTORE DATABASE** コマンドの後にカタログ・パーティションから発行する単一の **UPGRADE DATABASE** コマンドを続けて、データベースを正常にアップグレードすることができます。
- パーティション・データベース環境では、1 つの表スペースが、異なるデータベース・パーティション間で異なるストレージ・グループに関連付けられる可能性があります。リダイレクト・リストアで表スペース・コンテナーが DMS から自動ストレージに変更されると、その表スペースはデフォルトのストレージ・グループに関連付けられます。異なるデータベース・パーティションに対してリダイレクト・リストアが行われてから次回また行われるまでの間に新しいデフォルト・ストレージ・グループが選択された場合には、その表スペースのストレージ・グループの関連付けは、パーティション・データベース環境において矛盾することになります。このような場合、必要であれば、ALTER TABLESPACE ステートメントを使用して、すべてのデータベース・パーティションの表スペースで自動ストレージを使用するように変更し、リバランスを行ってください。
- **TRANSPORT** オプションは、クライアントとデータベースのコード・ページが同じである場合にのみサポートされます。

スナップショット・リストア

従来の (スナップショット以外の) リストアのように、スナップショット・バックアップ・イメージをリストアする時のデフォルトの動作は、ログ・ディレクトリーをリストアしない、**LOGTARGET EXCLUDE** です。

いずれかのログ・ディレクトリーのグループ ID がリストアする他のパスのいずれかと共有されていることが DB2 データベース・マネージャーにより検出された場合、エラーが返されます。この場合、ログ・ディレクトリーが

リストアに含まれる必要があるため、**LOGTARGET INCLUDE** または **LOGTARGET INCLUDE FORCE** を指定する必要があります。

DB2 データベース・マネージャーは、バックアップ・イメージからのパスのリストアが行われる前に既存のログ・ディレクトリー (1 次、ミラー、およびオーバーフロー) を保存するために、すべての方法を試みます。

ログ・ディレクトリーをリストアする場合、ディスク上に事前に存在するログ・ディレクトリーがバックアップ・イメージ中のログ・ディレクトリーと競合することが DB2 データベース・マネージャーによって検出されたなら、DB2 データベース・マネージャーによってエラーが報告されます。その場合、**LOGTARGET INCLUDE FORCE** を指定した場合にはこのエラーは抑止され、イメージのログ・ディレクトリーがリストアされて、以前に存在していたログ・ディレクトリーはすべて削除されます。

特殊なケースとして **LOGTARGET EXCLUDE** オプションが指定されていて、ログ・ディレクトリーのパスがデータベース・ディレクトリー (例えば、/NODE*xxxx*/SQL*xxxxx*/LOGSTREAM*xxxxx*/) の下にあるという場合があります。この場合は、リストアによりログ・ディレクトリーはデータベース・パスとして上書きされ、その下位にあるすべての内容はリストアされます。このシナリオに該当することが DB2 データベース・マネージャーによって検出された場合、そのログ・ディレクトリー中にログ・ファイルが存在しているなら、エラーが報告されます。 **LOGTARGET EXCLUDE FORCE** を指定した場合には、このエラーは抑止され、ディスク上の矛盾するログ・ディレクトリーは、バックアップ・イメージのそれらのログ・ディレクトリーで上書きされます。

表スペースとスキーマの転送

表スペースとスキーマの完全なリストを指定する必要があります。

転送時にターゲット・データベースがアクティブでなければなりません。

オンライン・バックアップ・イメージを使用する場合、ステージング・データベースはバックアップの最後までロールフォワードされます。オフライン・バックアップ・イメージを使用する場合、ロールフォワード処理は行われません。

dftdbpath データベース・パラメーターで指定したパスの下に、バックアップ・イメージに含まれるシステム・カタログ表スペースから成るステージング・データベースが作成されます。このデータベースは、**RESTORE DATABASE** コマンドが完了した時点でドロップされます。ステージング・データベースは、転送される表スペース内のオブジェクトを再生成するために使用される DDL を抽出するのに必要となります。

表スペースの転送時、DB2 データベース・マネージャーはページ・サイズが一致する最初の使用可能バッファー・プールを、転送される表スペースに割り当てようとします。転送される表スペースとページ・サイズが一致するバッファー・プールがターゲット・データベースにない場合は、隠しバッファー・プールが割り当てられることがあります。隠しバッファー・プールとは、転送される表スペース用の一時的なプレースホルダーのことです。転送完了後に、転送された表スペースに割り当てられたバッファー・プールを確認することができます。 **ALTER TABLESPACE** コマンドを発行することによって、バッファー・プールを更新できます。

データベースのロールフォワードで表スペース・スキーマ転送ログ・レコードが検出された場合、対応する転送済み表スペースはオフラインになってドロップ・ペンディング状態に変わります。これは、転送された表スペースとその内容を再作成するための転送済み表スペースの完全なログが、データベースにないためです。転送完了後にターゲット・データベースのフルバックアップを取ることができるので、その後のロールフォワードがログ・ストリーム内のスキーマ転送のポイントを通過することはありません。

ストレージ・グループの転送

転送操作では、ターゲット・データベースに現在定義されているストレージ・グループを変更することはできません。また、転送時に新規ストレージ・グループを明示的に作成することはできません。

転送のデフォルトのターゲット・ストレージ・グループは、操作のターゲット・データベースのデフォルトのストレージ・グループです。転送操作中にリストアするすべての表スペースを、ターゲット・データベースの特定のストレージ・グループに明示的にリダイレクトすることもできます。

転送操作中、**RESTORE** コマンドに **TRANSPORT REDIRECT** オプションを使用して実行すると、自動ストレージ表スペースのデフォルトのストレージ・グループ構成は、バックアップ・イメージで設定されている構成ではなく、ターゲット・データベースのストレージ・グループおよびストレージ・グループ・パスになります。なぜなら、自動ストレージ表スペースは、ターゲット・データベースの定義に従い、既存のストレージ・グループ・パスにリストアして直接リダイレクトする必要があるからです。

サスペンド入出力とオンライン・スプリット・ミラー・サポートによる高可用性

IBM DB2 サーバーのサスペンド入出力サポートにより、データベースをオフラインにしないで 1 次データベースのスプリット・ミラー・コピーを行うことができます。これを使用すると、1 次データベースで障害が発生した場合にテークオーバーするスタンバイ・データベースを非常に短い時間で作成することができます。

ディスク・ミラーリングは、データを 2 つの異なるハード・ディスクに同時に書き込む処理です。データの一方のコピーは、他方のミラーと呼ばれます。ミラーの分割とは、2 つのコピーを分離する処理のことです。

ディスク・ミラーリングを使用することにより、1 次データベースの 2 次コピーを保持することができます。 DB2 サーバーのサスペンド入出力機能を使用すると、データベースをオフラインにしないでデータベースの 1 次ミラー・コピーと 2 次ミラー・コピーを分割することができます。 1 次データベース・コピーと 2 次データベース・コピーが分割されると、2 次データベースは 1 次データベースで障害が起きた場合に操作をテークオーバーできるようになります。

大きなデータベースは DB2 サーバー・バックアップ・ユーティリティーを使用してバックアップしないという場合、サスペンド入出力およびスプリット・ミラー機能を使用してミラー・イメージからコピーを作成することができます。この方法では以下の利点もあります。

- 稼働マシンからバックアップ操作のオーバーヘッドを除去します。

- 高速にシステムを複製します。
- アイドル・スタンバイ・フェイルオーバーを高速にインプリメントできます。初期のリストア操作は不要です。また、ロールフォワードが遅すぎたりエラーが発生する場合に、再初期化を高速に実行してこれらに対応することが可能です。

db2inidb コマンドは、スプリット・ミラーを初期化して以下のように使用できるようにします。

- クローン・データベースとして
- スタンバイ・データベースとして
- バックアップ・イメージとして

このコマンドは、スプリット・ミラーに対してのみ発行することができます。そのコマンドは、スプリット・ミラーを使用する前に実行しなければなりません。

パーティション・データベース環境では、入出力を中断してすべてのデータベース・パーティションに同時に書き込む必要はありません。 1 つ以上のデータベース・パーティションのサブセットを中断して、オフライン・バックアップを実行するためにスプリット・ミラーを作成できます。カタログ・パーティションがサブセットに含まれる場合は、そのパーティションを最後に中断するデータベース・パーティションにする必要があります。

パーティション・データベース環境では、**db2inidb** コマンドは各データベース・パーティション上でそれぞれ実行する必要があります。それから、それらのデータベース・パーティションのスプリット・イメージを使用することができます。db2inidb コマンドは、**db2_all** コマンドを使用してすべてのデータベース・パーティションで同時に実行することができます。しかし、RELOCATE USING オプションを使用する場合は、**db2_all** コマンドを使用して全データベース・パーティションに対して同時に **db2inidb** を実行することはできません。データベース・パーティションごとにそれぞれ別個の構成ファイル (変更するデータベース・パーティションの NODENUM 値が含まれる) を用意する必要があります。例えば、データベースの名前を変更する場合は、すべてのデータベース・パーティションが影響を受けることになり、各データベース・パーティションごとに別個の構成ファイルを用意して **db2relocatedb** コマンドを実行する必要があります。単一データベース・パーティションに属するコンテナーを移動する場合は、そのデータベース・パーティションに対して一度だけ **db2relocatedb** コマンドを実行することが必要です。

注: スプリット・ミラーが、データベースを構成するすべてのコンテナーおよびディレクトリー (ボリューム・ディレクトリーを含む) を含んでいることを確認してください。この情報を収集するには、DBPATHS 管理ビューを参照してください。このビューは、分割する必要のあるデータベースのすべてのファイルとディレクトリーを表示します。

db2inidb - ミラーリングされたデータベースの初期化

スプリット・ミラー環境のミラーリングされたデータベースを初期化します。ミラーリングされたデータベースは、ロールフォワード・ペンディング状態にある 1 次データベースの複製として初期化したり、1 次データベースをリストアするためのバックアップ・イメージとして使用できます。

スプリット・ミラー・データベースを使用できるようにするには、まずこのコマンドを発行する必要があります。

許可

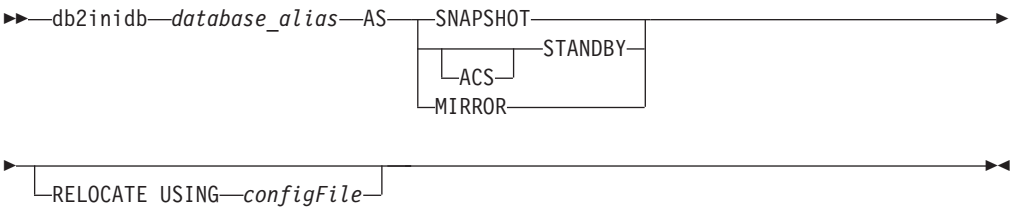
以下の権限のいずれか。

- SYSADM
- SYSCTRL
- SYSMAINT

必要な接続

なし

コマンド構文



コマンド・パラメーター

database_alias
初期設定するデータベースの別名を指定します。

SNAPSHOT
ミラーリングされたデータベースは、1 次データベースの複製として初期化されることを指定します。

STANDBY
データベースをロールフォワード・ペンディング状態にすることを指定します。 1 次データベースから新しいログ・ファイルをフェッチ、スタンバイ・データベースに適用することが可能です。スタンバイ・データベースは、1 次データベースがダウンした場合に、その代わりに使用できます。

ACS
db2inidb コマンドを、データベースの ACS スナップショット・コピーに対して使用して **STANDBY** アクションを実行するように指定します。このオプションが必要となる理由は、**db2inidb** コマンドを発行できる対象が **SET WRITE SUSPEND | RESUME** コマンドによって作成されたスプリット・ミラー・データベース・スナップショットに限定されているためです。

また、**ACS STANDBY** オプションを使用すると ACS スナップショット・コピーが開始されてロールフォワード・ペンディング状態になるので、DB2 **BACKUP** コマンドをそのスナップショット・イメージに対して正常に発行できます。このようにしないと、スナップショット・イメージに

接続しようとするとデータベースのコピーが RESTORE_PENDING 状態になり、将来リカバリーする際にバックアップ・コピーとして使用できなくなってしまいます。

このフィーチャーは、ACS スナップショットに基づいて DB2 オフロード・バックアップを作成するために、IBM Tivoli Storage FlashCopy® Manager などのストレージ・マネージャーと連携するという目的で特に導入されました。その他の目的のためにこのオプションを使用し、ACS スナップショット・コピーの内容をマウントしたり変更したりすると、DB2 バックアップを作成する場合であっても、将来に予期しない動作を引き起こす恐れがあります。

MIRROR ミラー・データベースを、1 次データベースのリストアに使用できるバックアップ・イメージにすることを指定します。

RELOCATE USING *configFile*
データベースがスナップショット、スタンバイ、またはミラーとして初期化される前に、*configFile* にリストされた情報に基づいてデータベース・ファイルを再配置することを指定します。 *configFile* の形式については、 208 ページの『db2relocatedb - データベースの再配置』を参照してください。

使用上の注意

db2inidb*database_alias* as mirror コマンドを発行する前に db2 connect to*database-alias* 操作を発行しないでください。初期化する前にスプリット・ミラー・データベースに接続すると、ロールフォワード・リカバリーで必要なログ・ファイルが消去されてしまいます。その接続によって、データベースは、以前にそのデータベースを中断した時点の状態に戻ります。中断の時点でデータベースが整合とマーク付けされると、DB2 データベース・システムはクラッシュ・リカバリーの必要はないと判断して、将来の利用のためにログを空にします。ログが空になった場合、ロールフォワードしようとすると、SQL4970N エラー・メッセージが戻されます。

パーティション・データベース環境では、すべてのデータベース・パーティションに対して **db2inidb** コマンドを発行する必要があります。その後、任意のデータベース・パーティションのスプリット・ミラーを使用できるようになります。**db2_all** コマンドを使用すれば、すべてのデータベース・パーティションに対して同時に **db2inidb** を実行することができます。

しかし、**RELOCATE USING** オプションを使用する場合は、 **db2_all** コマンドを使用して全パーティションに対して同時に **db2inidb** を実行することはできません。パーティションごとにそれぞれ別個の構成ファイル (変更対象のデータベース・パーティションの NODENUM 値が含まれる) を用意する必要があります。例えば、データベースの名前を変更する場合は、すべてのデータベース・パーティションが影響を受けることになり、各データベース・パーティションごとに別個の構成ファイルを用意して **db2relocatedb** コマンドを実行する必要があります。単一データベース・パーティションに属するコンテナーを移動する場合は、そのデータベース・パーティションに対して一度だけ **db2relocatedb** コマンドを実行することが必要です。

RELOCATE USING *configFile* パラメーターが指定されており、データベースの再配置が正常に実行されたなら、指定された *configFile* はデータベース・ディレクトリーにコピーされ、その名前が db2path.cfg に変更されます。それ以降のクラッシュ・リカバリーまたはロールフォワード・リカバリーにおいて、このファイルは、ログ・ファイルの処理時にコンテナー・パスの名前を変更するために使用されます。

クローン・データベースを初期化している場合、指定された *configFile* は、クラッシュ・リカバリー完了後にデータベース・ディレクトリーから自動的に除去されます。

スタンバイ・データベースまたはミラーリングされたデータベースを初期化している場合、指定された *configFile* ファイルは、ロールフォワード・リカバリーの完了後またはキャンセル後に、データベース・ディレクトリーから自動的に除去されます。 **db2inidb** 実行後には、新しいコンテナー・パスを db2path.cfg ファイルに追加できます。元のデータベースに対して CREATE 操作または ALTER TABLESPACE 操作を実行し、スタンバイ・データベース上で異なるパスを使用しなければならない場合には、このことが必要になります。

HADR 1 次またはスタンバイから取られたスプリット・ミラー・データベースの初期化を実行する際には、以下のいずれかが当てはまる場合、**STANDBY** パラメーターを使用してください。

- 新しいデータベースは HADR ペア内で動作する予定で、新しいペアの HADR 構成設定が元のペアの設定と同じではない。
- データベースをスタンドアロン・データベースとして初期化する予定である。

DB2 pureScale環境では、いずれかのメンバーから **db2inidb** コマンドを発行でき、コマンドを 1 回だけ発行する必要があります。

db2relocatedb - データベースの再配置

このコマンドは、ユーザー提供の構成ファイルで指定されたとおりに、データベースを名前変更したり、データベースやデータベースの一部 (コンテナー、ログ・ディレクトリーなど) を再配置します。このツールは、DB2 インスタンスおよびデータベース・サポート・ファイルに、必要な変更を行います。

ターゲット・データベースをオフラインにしてからでなければ、**db2relocatedb** コマンドを実行してターゲット・データベースの制御ファイルとメタデータを変更してはなりません。

db2relocatedb コマンドがデータベースのファイルおよび制御構造に加える変更は、ログに記録されないため、リカバリー不能です。特に、保存されているログ・ファイルによってデータベースがリカバリー可能な場合、データベースに対するコマンドの実行後にフルバックアップを行うのは良い習慣です。

許可

なし

前提条件

このコマンドを使用する前に、次のコマンドを実行してファイルを移動しておく必要があります。

```
mv /home/db2inst1/db2inst1/NODE0000/X /home/db2inst1/db2inst1/NODE0000/Y
```

ここで、*X* は古いデータベース名を、*Y* は新しいデータベース名を表します。

この追加のステップを実行して、コマンド **db2relocatedb** が確実にエラー・メッセージなしで実行されるようにする必要があります。このステップは、DB2 バージョン 10.1 以降でのみ必要です。

コマンド構文

コマンド・パラメーター

-f *configFilename*

データベースの再配置に必要な構成情報の入ったファイルの名前を指定します。これは、相対ファイル名でも絶対ファイル名でも構いません。構成ファイルのフォーマットは以下のとおりです。

```
DB_NAME=oldName,newName
DB_PATH=oldPath,newPath
INSTANCE=oldInst,newInst
NODENUM=nodeNumber
LOG_DIR=oldDirPath,newDirPath
CONT_PATH=oldContPath1,newContPath1
CONT_PATH=oldContPath2,newContPath2
...
STORAGE_PATH=oldStoragePath1,newStoragePath1
STORAGE_PATH=oldStoragePath2,newStoragePath2
...
FAILARCHIVE_PATH=newDirPath
LOGARCHMETH1=newDirPath
LOGARCHMETH2=newDirPath
MIRRORLOG_PATH=newDirPath
OVERFLOWLOG_PATH=newDirPath
...
```

ここで、

DB_NAME

再配置されるデータベースの名前を指定します。データベース名を変更する場合は、古い名前と新規の名前の両方を指定する必要があります。このフィールドは必須です。

DB_PATH

再配置されるデータベースの元のパスを指定します。データベース・パスが変更される場合、古いパスと新規のパスの両方を指定する必要があります。このフィールドは必須です。

INSTANCE

データベースが存在する場所のインスタンスを指定します。データ

ベースが新規のインスタンスに移動される場合、古いインスタンスと新規のインスタンスの両方を指定する必要があります。このフィールドは必須です。

NODENUM
変更されるデータベース・ノードのノード番号を指定します。デフォルトは 0 です。

LOG_DIR
ログ・パスのロケーション内の変更を指定します。ログ・パスが変更される場合、古いパスと新しいパスの両方を指定する必要があります。ログ・パスがデータベース・パスの下にある場合、パスは自動的に更新されるので、この指定はオプションです。

CONT_PATH
表スペース・コンテナーのロケーション内の変更を指定します。古いコンテナー・パスと新規のコンテナー・パスの両方を指定する必要があります。複数のコンテナー・パスを変更する場合、複数の **CONT_PATH** 行を指定できます。コンテナー・パスがデータベース・パスの下にある場合、パスは自動的に更新されるので、この指定はオプションです。同じ古いパスが共通の新規パスで置換される場所で、複数のコンテナーに変更を行う場合、単一の **CONT_PATH** 項目が使用されます。このような場合、古いパス、新規パスの両方にアスタリスク (*) をワイルドカードとして使用できます。

STORAGE_PATH
データベースのいずれかのストレージ・パスの場所を変更することを指定します。古いストレージ・パスと新しいストレージ・パスの両方を指定する必要があります。複数のストレージ・パスを変更する場合、複数の **STORAGE_PATH** 行を指定できます。このパラメーターを指定して、すべてのストレージ・グループ内の任意のストレージ・パスを変更することができます。ただし、個々のストレージ・グループのストレージ・パスを変更するために、このパラメーターを指定することはできません。

注: このパラメーターは、AUTOMATIC STORAGE NO 節を指定して作成されたデータベースには適用されません。 AUTOMATIC STORAGE NO 節を指定してデータベースを作成できますが、AUTOMATIC STORAGE 節は非推奨になり、将来のリリースで除去される可能性があります。

FAILARCHIVE_PATH
データベース・マネージャーが 1 次アーカイブ・ロケーションにも 2 次アーカイブ・ロケーションにもログ・ファイルをアーカイブできない場合に、ログ・ファイルをアーカイブする新しいロケーションを指定します。このフィールドは、再配置されるデータベースに **failarchpath** 構成パラメーターが設定されている場合のみ指定してください。

LOGARCHMETH1
新しい 1 次アーカイブ・ロケーションを指定します。このフィール

ドは、再配置されるデータベースに **logarchmeth1** 構成パラメーターが設定されている場合のみ指定してください。

LOGARCHMETH2
新しい 2 次アーカイブ・ロケーションを指定します。このフィールドは、再配置されるデータベースに **logarchmeth2** 構成パラメーターが設定されている場合のみ指定してください。

MIRRORLOG_PATH
ミラー・ログ・パスの新しいロケーションを指定します。ストリングがパス名を指していなければなりません。また、相対パス名ではなく、絶対パス名でなければなりません。このフィールドは、再配置されるデータベースに **mirrorlogpath** 構成パラメーターが設定されている場合のみ指定してください。

OVERFLOWLOG_PATH
ロールフォワード操作に必要なログ・ファイルの検索、アーカイブから取り出されたアクティブ・ログ・ファイルの保管、db2ReadLog API が必要とするログ・ファイルの検索と保管を行う新しいロケーションを指定します。このフィールドは、再配置されるデータベースに **overflowlogpath** 構成パラメーターが設定されている場合のみ指定してください。

ブランク行またはコメント文字 (#) で始まる行は無視されます。

例

例 1

データベース TESTDB の名前を PRODDB に、パス /home/db2inst1 にあるインスタンス db2inst1 で変更するには、以下の構成ファイルを作成します。

```
DB_NAME=TESTDB,PRODDB
DB_PATH=/home/db2inst1
INSTANCE=db2inst1
NODENUM=0
```

構成ファイルが作成されたら、すべての自動ストレージ・パスを、新規データベース名と一致するように変更する必要があります。

```
rename /home/db2inst1/db2inst1/TESTDB /home/db2inst1/db2inst1/PRODDB
```

構成ファイルを relocate.cfg として保存し、以下のコマンドを使用して、データベース・ファイルへの変更を行います。

```
db2relocatedb -f relocate.cfg
```

例 2

データベース DATAB1 をパス /dbpath のインスタンス jsmith からインスタンス prodinst に移動するには、以下のようにします。

1. ディレクトリー /dbpath/jsmith 内のファイルを /dbpath/prodinst に移動します。
2. 以下の構成ファイルと **db2relocatedb** コマンドを使用して、データベース・ファイルに変更を行います。

```
DB_NAME=DATAB1
DB_PATH=/dbpath
INSTANCE=jsmith,prodinst
NODENUM=0
```

例 3

パス /databases/PRODDB のインスタンス inst1 内に存在するデータベース PRODDB です。 2 つの表スペース・コンテナーのロケーションを、以下のように変更する必要があります。

- SMS コンテナー /data/SMS1 を /DATA/NewSMS1 に移動する必要があります。
- DMS コンテナー /data/DMS1 を /DATA/DMS1 に移動する必要があります。

物理ディレクトリーおよびファイルが、新規のロケーションに移動された後で、新規のロケーションを認識するように、以下の構成ファイルと **db2relocatedb** コマンドを使用して、データベース・ファイルに変更を行います。

```
DB_NAME=PRODDB
DB_PATH=/databases/PRODDB
INSTANCE=inst1
NODENUM=0
CONT_PATH=/data/SMS1,/DATA/NewSMS1
CONT_PATH=/data/DMS1,/DATA/DMS1
```

例 4

インスタンス db2inst1 に存在するデータベース TESTDB は、パス /databases/TESTDB に作成されました。表スペースは、以下のコンテナーと共に作成されました。

```
TS1
TS2_Cont0
TS2_Cont1
/databases/TESTDB/TS3_Cont0
/databases/TESTDB/TS4/Cont0
/Data/TS5_Cont0
/dev/rTS5_Cont1
```

TESTDB は新規システムに移動されます。新規システムのインスタンスは newinst になり、データベースのロケーションは /DB2 になります。

データベースを移動する場合、/databases/TESTDB/db2inst1 ディレクトリーに存在するすべてのファイルは、/DB2/newinst ディレクトリーに移動する必要があります。これは、最初の 5 つのコンテナーが、この移動の一部として再配置されることを意味します。(最初の 3 つはデータベース・ディレクトリーに相対で、次の 2 つはデータベース・パスに相対です。) これらのコンテナーがデータベース・ディレクトリーまたはデータベース・パス内にあるため、構成ファイルにリストする必要はありません。 2 つの残りのコンテナーが新規システムで異なるロケーションに移動された場合は、構成ファイルにリストする必要があります。

物理ディレクトリーおよびファイルが新規のロケーションに移動された後で、新規のロケーションを認識するように、以下の構成ファイルと **db2relocatedb** を使用して、データベース・ファイルに変更を行います。

```
DB_NAME=TESTDB
DB_PATH=/databases/TESTDB,/DB2
INSTANCE=db2inst1,newinst
NODENUM=0
CONT_PATH=/Data/TS5_Cont0,/DB2/TESTDB/TS5_Cont0
CONT_PATH=/dev/rTS5_Cont1,/dev/rTESTDB_TS5_Cont1
```

例 5

データベース TESTDB には、データベース・パーティション・サーバー 10 および 20 に 2 つのデータベース・パーティションがあります。このインスタンスは servinst で、データベース・パスは両方のデータベース・パーティション・サーバーで /home/servinst です。データベースの名前は SERVDB に変更され、データベース・パスは両方のデータベース・パーティション・サーバーで /databases に変更されます。さらに、ログ・ディレクトリーはデータベース・パーティション・サーバー 20 で、/testdb_logdir から /servdb_logdir に変更されます。

両方のデータベース・パーティションに変更が行われているため、構成ファイルは各データベース・パーティションに作成され、**db2relocatedb** は対応する構成ファイルを使用する各データベース・パーティション・サーバーで実行される必要があります。

データベース・パーティション・サーバー 10 では、以下の構成ファイルが使用されます。

```
DB_NAME=TESTDB,SERVDB
DB_PATH=/home/servinst,/databases
INSTANCE=servinst
NODENUM=10
```

データベース・パーティション・サーバー 20 では、以下の構成ファイルが使用されます。

```
DB_NAME=TESTDB,SERVDB
DB_PATH=/home/servinst,/databases
INSTANCE=servinst
NODENUM=20
LOG_DIR=/testdb_logdir,/servdb_logdir
```

例 6

パス /home/maininst のインスタンス maininst 内に存在するデータベース MAINDB です。 4 つの表スペース・コンテナーのロケーションを、以下のように変更する必要があります。

```
/maininst_files/allconts/C0 needs to be moved to /MAINDB/C0
/maininst_files/allconts/C1 needs to be moved to /MAINDB/C1
/maininst_files/allconts/C2 needs to be moved to /MAINDB/C2
/maininst_files/allconts/C3 needs to be moved to /MAINDB/C3
```

物理ディレクトリーおよびファイルが、新規のロケーションに移動された後で、新規のロケーションを認識するように、以下の構成ファイルと **db2relocatedb** コマンドを使用して、データベース・ファイルに変更を行います。

同様の変更が、すべてのコンテナーに対して行われました。すなわち、/maininst_files/allconts/ が /MAINDB/ で置換され、ワイルドカード文字のある単一項目が使用できるようになります。

```
DB_NAME=MAINDB
DB_PATH=/home/maininst
INSTANCE=maininst
NODENUM=0
CONT_PATH=/maininst_files/allconts/*, /MAINDB/*
```

使用上の注意

データベースが属するインスタンスを変更する場合、インスタンスおよびデータベース・サポート・ファイルに確実に変更が加えられるようにするため、このコマンドを実行する前に以下の事柄を行う必要があります。

- データベースが他のインスタンスに移動されている場合は、新規のインスタンスを作成します。新規インスタンスは、現在データベースがあるインスタンスと同じリリース・レベルでなければなりません。
- 新しいインスタンスの所有者が現在のインスタンスの所有者でない場合は、新しいインスタンスの所有者へアクセス権を付与します。
- 新規インスタンスが常駐するシステムにコピーされるデータベースに属するファイルとデバイスをコピーします。パス名は必要に応じて変更する必要があります。ただし、データベース・ファイルの移動先のディレクトリー内にデータベースが既にある場合、不用意に既存の sqldbdir ファイルを上書きしてしまって、既存のデータベースへの参照を除去する可能性があります。そのような事態になった場合、 **db2relocatedb** ユーティリティーを使用することはできません。その場合、**db2relocatedb** の代わりにリダイレクト・リストア操作を使用できます。
- インスタンス所有者に所有されるように、コピーされたファイル/デバイスのアクセス権を変更します。

複数のデータベースがあるデータベース・パスからデータベースを移動する場合には、そのデータベース・パス内の sqldbdir ディレクトリーを移動するのではなくコピーする必要があります。このディレクトリーは、移動されないデータベースを DB2 が見つけるために、そのまま古い場所に置いておくことが必要です。sqldbdir ディレクトリーを新しい場所にコピーした後、LIST DB DIRECTORY ON *newPath* コマンドを実行すると、移動されなかったデータベースがリスト表示されます。これらの参照は削除できず、その名前を持つ新しいデータベースをこの同じパス上に作成することもできません。ただし、その名前を持つデータベースを別のパスに作成することはできます。

ALTER TABLESPACE MANAGED BY AUTOMATIC STORAGE ステートメントを用いて自動ストレージを使用するように変換された、表スペースの既存のユーザー作成コンテナーを移動するために、**db2relocatedb** コマンドを使用することはできません。

インスタンスが変更されている場合、コマンドは新規のインスタンス所有者によって実行される必要があります。

パーティション・データベース環境では、変更が必要なすべてのデータベース・パーティションに対してこのツールを実行する必要があります。データベース・パーティションごとにそれぞれ別個の構成ファイル (変更対象のデータベース・パーティションの NODENUM 値が含まれる) を用意する必要があります。例えば、データベースの名前を変更する場合は、すべてのデータベース・パーティションが影響を受けることになり、各データベース・パーティションごとに別個の構成ファイルを用

意して **db2relocatedb** コマンドを実行する必要があります。単一データベース・パーティションに属するコンテナーを移動する場合は、そのデータベース・パーティションに対して一度だけ **db2relocatedb** コマンドを実行することが必要です。

db2relocatedb コマンドを使用して、ロードが進行中のデータベースや、**LOAD RESTART** または **LOAD TERMINATE** コマンドの完了を待機しているデータベースを再配置することはできません。

制約事項: パーティション・データベース環境では、同じ装置に常駐する複数のロジカル・パーティションの 1 つであるノードの全体を再配置することはできません。

db2look - DB2 統計および DDL 抽出ツール

テスト・データベース上に実動データベースのデータベース・オブジェクトを再作成するのに必要な、データ定義言語 (DDL) ステートメントを抽出します。 **db2look** コマンドは、オブジェクト・タイプごとに DDL ステートメントを生成します。 なお、このコマンドは、SYSTOOLS スキーマのすべてのオブジェクトを無視します。ただし、ユーザー定義関数とストアード・プロシージャーについては例外です。

テスト・システムに実動システムのデータのサブセットを含めておくと、便利なことが多くあります。しかし、そのようなテスト・システム用に選択したアクセス・プランが、必ずしも実動システム用に選択したアクセス・プランと同じであるとは限りません。ただし、**db2look** ツールを使用すると、実動システムで使用するものとアクセス・プランが類似しているテスト・システムを作成することができます。このツールを使用して、テスト・データベース上の実動データベースにあるオブジェクトのカタログ統計を複製するのに必要な UPDATE ステートメントを生成することができます。また、このツールを使用して、**UPDATE DATABASE CONFIGURATION**、**UPDATE DATABASE MANAGER CONFIGURATION**、および **db2set** コマンドを生成することで、テスト・データベース上の照会オプティマイザー関連の構成パラメーターとレジストリー変数の値を、実動データベースでの値に合うように設定できます。

db2look コマンドによって生成される DDL ステートメントは、元の SQL オブジェクトのすべての特性を複製するとは限らないため、生成された DDL ステートメントを必ず確認するようにしてください。パーティション・データベース環境の表スペースでは、アクティブでないデータベース・パーティションが存在する場合、DDL が完全でない可能性があります。 **ACTIVATE DATABASE** コマンドを使用することによって、すべてのデータベース・パーティションがアクティブであることを確認してください。

許可

システム・カタログ表に対する SELECT 特権。

表スペース・コンテナー DDL を生成する場合などでは、以下の権限のいずれか 1 つが必要です。

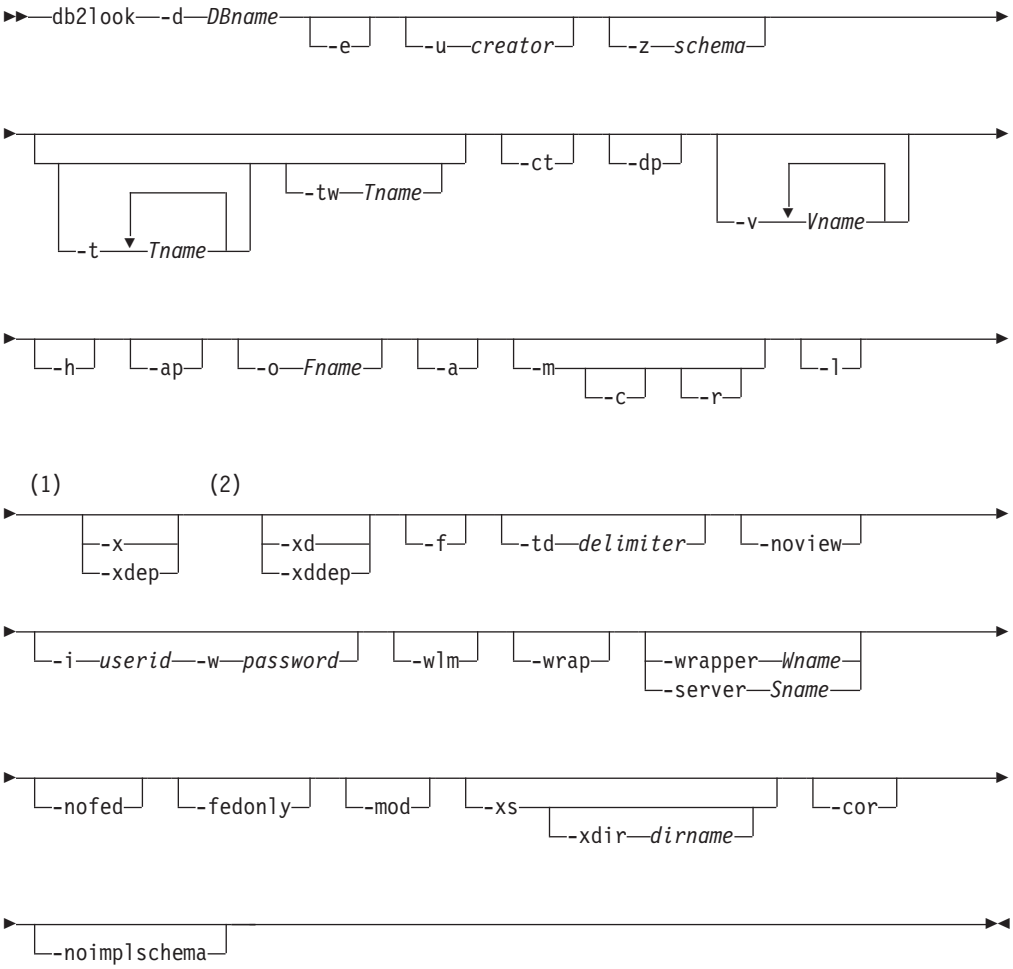
- SYSADM
- SYSCTRL
- SYSMAINT
- SYSMON

- DBADM
- ADMIN_GET_STORAGE_PATHS 表関数に対する EXECUTE 特権

必要な接続

なし

コマンド構文



注:

1 **-x** パラメーターと **-xdep** パラメーターを両方とも指定することはできません。

2 **-xd** パラメーターと **-xddep** パラメーターを両方とも指定することはできません。

コマンド・パラメーター

-d *DBname*

照会する実動データベースの別名。 *DBname* としては、DB2 for Linux, UNIX, and Windows または DB2 Version 9.1 for z/OS データベースの名前

を指定できます。*DBname* が DB2 for z/OS データベースである場合には、**db2look** コマンドは、以下のステートメントを OS/390® および z/OS オブジェクト用に生成します。

- 表、索引、ビュー、およびユーザー定義特殊タイプ用の DDL ステートメント
- 表、列、列分散および索引用の UPDATE 統計ステートメント

これらの DDL および UPDATE 統計は、DB2 for Linux, UNIX, and Windows データベースには適用できますが、DB2 for z/OS データベースには適用できません。これらのステートメントは、OS/390 および z/OS のオブジェクトを抽出して、それらを DB2 for Linux, UNIX, and Windows データベース内に再作成しようとするユーザーに役立ちます。

-e 以下のデータベース・オブジェクト用の DDL ステートメントを抽出します。

- 別名
- 監査ポリシー
- チェック制約
- 関数マッピング
- 関数テンプレート
- グローバル変数
- 索引 (パーティション表上のパーティション索引を含む)
- SPECIFICATION ONLY 指定の索引
- マテリアライズ照会表 (MQT)
- ニックネーム
- 主キー、参照整合性、およびチェック制約
- 参照整合性制約
- ロール
- スキーマ
- セキュリティー・ラベル
- セキュリティー・ラベル・コンポーネント
- セキュリティー・ポリシー
- 順序
- サーバー
- ストアード・プロシージャー
- 表

  注: SYSIBM.SYSTABLES カタログ表の列 STATISTICS_PROFILE の値は含まれません。
- トリガー
- トラステッド・コンテキスト
- タイプ・マッピング
- ユーザー・マッピング
- ユーザー定義特殊タイプ

- ユーザー定義関数
- ユーザー定義メソッド
- ユーザー定義構造化タイプ
- ユーザー定義トランスフォーム
- ビュー
- ラッパー

db2look コマンドによって生成された DDL ステートメントを使用してユーザー定義関数を再作成する場合、その関数を使用できる状態にするには、関数が参照するソース・コード (EXTERNAL NAME 節など) が使用可能でなければなりません。

-u *creator*

指定された作成者 ID で作成されたオブジェクトに対する DDL ステートメントを生成します。出力を、指定された作成者 ID で作成されたオブジェクトに制限します。出力に作動不能オブジェクトは含まれません。作動不能オブジェクトを表示するには、**-a** パラメーターを使用します。 **-a** パラメーターを指定した場合、**-u** パラメーターは無視されます。

-z *schema*

指定されたスキーマ名を持つオブジェクトに対する DDL ステートメントを生成します。出力を、指定されたスキーマ名を持つオブジェクトに制限します。出力に作動不能オブジェクトは含まれません。作動不能オブジェクトを表示するには、**-a** パラメーターを使用します。 **-z** パラメーターを指定しない場合には、すべてのスキーマ名のオブジェクトが抽出されます。 **-a** パラメーターを指定した場合、**-z** パラメーターは無視されます。また、このパラメーターも、フェデレーテッド DDL ステートメントでは無視されます。

-t *Tname1 Tname2 ... TnameN*

指定した表とその従属オブジェクトに対する DDL ステートメントを生成します。表リストに指定された表だけに出力を制限し、ユーザー指定のすべての表の、すべての従属オブジェクトについて、DDL ステートメントを生成します。表の最大数は 30 です。リストを以下のように指定します。

- 表名をブランク・スペースで区切ります。
- 大/小文字の区別のある名前と 2 バイト文字セット (DBCS) 名は、円記号 (¥) および二重引用符 (" ") で囲む必要があります (例えば ¥" MyTabLe ¥")。
- 複数語の表名は、円記号と 2 セットの二重引用符 (例えば "¥"My Table¥"") で囲むことにより、その対が一語ごとにコマンド行プロセッサー (CLP) で評価されないようにします。 1 セットの二重引用符だけを使用した場合 (例えば "My Table") には、すべての語が大文字に変換され、 **db2look** コマンドは大文字の表名 (例えば MY TABLE) を探すことになります。

-t パラメーターを **-l** パラメーターと共に指定すると、パーティション表がサポートされます。

-z *schema* パラメーターを使用せずに表名を完全修飾するために、*schema.table* という形式の 2 部から成る表名を使用することができます。

表が持っている従属オブジェクトが、表のスキーマとは異なるスキーマにあるときに、その従属オブジェクトのために生成される DDL ステートメントを必要とする場合には、2 部から成る表名を使用してください。 **-z** *schema* パラメーターを使用してスキーマを指定する場合、そのパラメーターは同じ親スキーマを持たない従属オブジェクトを除外するので、そのような従属オブジェクトのための DDL ステートメントは生成されないことになります。

-tw *Tname*

Tname で指定するパターンに一致する名前を持つ表について DDL ステートメントを生成し、それらの表のすべての従属オブジェクトについて、DDL ステートメントを生成します。 *Tname* は、1 つの値だけでなければなりません。 *Tname* 内の下線文字 (_) は、任意の 1 文字を表します。パーセント記号 (%) は、ゼロ個以上の文字のストリングを表します。**-tw** を指定した場合は、**-t** オプションが無視されます。

-z *schema* パラメーターを使用せずに表名を完全修飾するために、*schema.table* という形式の 2 部から成る表名を使用することができます。表が持っている従属オブジェクトが、表のスキーマとは異なるスキーマにあるときに、その従属オブジェクトのために生成される DDL ステートメントを必要とする場合には、2 部から成る表名を使用してください。 **-z** *schema* パラメーターを使用してスキーマを指定する場合、そのパラメーターは同じ親スキーマを持たない従属オブジェクトを除外するので、そのような従属オブジェクトのための DDL ステートメントは生成されないことになります。

-ct オブジェクト作成時刻に基づいて DDL ステートメントを生成します。オブジェクト DDL ステートメントは、正しい従属順序では表示されない可能性があります。 **-ct** パラメーターを指定する場合、**db2look** コマンドは以下の追加パラメーターだけをサポートします: **-e**、**-a**、**-u**、**-z**、**-t**、**-tw**、**-v**、**-l**、**-noview**、および **-wlm**。 **-ct** パラメーターを、**-z** および **-t** パラメーターと共に使用した場合、**db2look** コマンドは、表、統計ビュー、列、および索引についての統計を複製するのに必要な UPDATE ステートメントを生成します。

-dp CREATE ステートメントの前に DROP ステートメントを生成します。ドロップされるオブジェクトに依存するオブジェクトが存在する場合、その DROP ステートメントは機能しない可能性があります。例えば、スキーマに依存する表がある場合には、そのスキーマをドロップすることはできません。また、ユーザー定義タイプまたはユーザー定義関数に依存するタイプ、関数、トリガー、または表がある場合、それらのユーザー定義タイプまたはユーザー定義関数をドロップすることはできません。型付き表の場合、DROP TABLE HIERARCHY ステートメントはルート表についてのみ生成されます。表がドロップされると索引、主キーと外部キー、および制約も常にドロップされるため、それらについての DROP ステートメントは生成されません。 RESTRICT ON DROP 属性を持つ表はドロップできません。

-v *Vname1 Vname2 ... VnameN*

指定したビューに対する DDL ステートメントを生成しますが、それらの従属オブジェクトに対しては生成しません。ビューの最大数は 30 です。大/小文字の区別、DBCS、および複数語表名を制御する規則は、ビュー名にも適用されます。**-t** パラメーターを指定した場合、**-v** パラメーターは無視されます。

ビューを完全修飾するために、*schema.view* フォーマットの 2 部から成るビュー名を使用できます。

-h ヘルプ情報を表示します。 このパラメーターを指定した場合、他のすべてのパラメーターは無視されます。

-ap 監査ポリシーを他のデータベース・オブジェクトに関連付けるのに必要な AUDIT USING ステートメントを生成します。

-o *Fname*
出力を *Fname* ファイルに書き込みます。拡張子を指定しない場合、拡張子 **.sql** が使用されます。このパラメーターを指定しない場合、出力は標準出力に書き込まれます。

-a 作動不能オブジェクトを含め、他のユーザーによって作成されたオブジェクトに対する DDL ステートメントを生成します。例えば、このパラメーターを **-e** パラメーターと共に指定した場合、データベース内のすべてのオブジェクト用の DDL ステートメントが抽出されます。このパラメーターを **-m** パラメーターと共に指定した場合には、データベース内のすべてのユーザー作成の表および索引用の UPDATE 統計ステートメントが抽出されます。

-u および **-a** パラメーターをいずれも指定しない場合には、USER 環境変数が使用されます。 UNIX オペレーティング・システム上では、この変数を明示的に設定する必要はありません。ただし、Windows オペレーティング・システム上では、USER 環境変数のデフォルト値がありません。したがって、SYSTEM 変数にユーザー変数を設定するか、またはセッションに対して **set USER=***username* コマンドを発行する必要があります。

-m 表、統計ビュー、列、および索引についての統計を複製するのに必要な UPDATE ステートメントを生成します。 **-m** パラメーターを使用することを、模倣モードでの実行といいます。

-c このオプションを指定した場合、**db2look** コマンドは、COMMIT、CONNECT、および CONNECT RESET ステートメントを生成しません。デフォルト・アクションでは、これらのステートメントを生成します。 **-m** または **-e** パラメーターも指定されているのでなければ、このオプションは無視されます。

-r このオプションを **-m** パラメーターと共に指定する場合、**db2look** コマンドは **RUNSTATS** コマンドを生成しません。デフォルト・アクションでは、**RUNSTATS** コマンドを生成します。

重要: 別のデータベースに対して、**-m** パラメーター付きの **db2look** コマンドを使用して作成されたコマンド・プロセッサー・スクリプトを実行する予定の場合 (例えば、テスト・データベースのカタログ統計を実動環境のカタログ統計と突き合わせる場合)、両方のデータベースが同じコード・セットとテリトリーを使用する必要があります。

-l 以下のデータベース・オブジェクト用の DDL ステートメントを生成します。

- ユーザー定義表スペース
- ユーザー定義ストレージ・グループ
- ユーザー定義データベース・パーティション・グループ

- ユーザー定義バッファー・プール

-x GRANT ステートメントなどの許可 DDL ステートメントを生成します。

サポートされている許可には、以下のものが含まれます。

- 列: UPDATE、REFERENCES
- データベース: ACCESSCTRL、BINDADD、CONNECT、CREATETAB、CREATE_EXTERNAL_ROUTINE、CREATE_NOT_FENCED_ROUTINE、DATAACCESS、DBADM、EXPLAIN、IMPLICIT_SCHEMA、LOAD、QUIESCE_CONNECT、SECADM、SQLADM、WLMADM
- 免除
- グローバル変数
- 索引: CONTROL
- パッケージ: CONTROL、BIND、EXECUTE
- ロール
- スキーマ: CREATEIN、DROPIN、ALTERIN
- セキュリティー・ラベル
- シーケンス: USAGE、ALTER
- ストアード・プロシージャー: EXECUTE
- 表: ALTER、SELECT、INSERT、DELETE、UPDATE、INDEX、REFERENCE、CONTROL
- ビュー: SELECT、INSERT、DELETE、UPDATE、CONTROL
- ユーザー定義関数 (UDF): EXECUTE
- ユーザー定義メソッド: EXECUTE
- 表スペース: USE
- ワークロード: USAGE

注: -t または **-tw** パラメーターのいずれかと共に使用した場合、このパラメーターは従属オブジェクトのための許可 DDL ステートメントを生成しません。親オブジェクトと従属オブジェクトに対して許可 DDL ステートメントを生成する場合は、**-xdep** パラメーターを使用してください。

-xdep 親オブジェクトと従属オブジェクトに対して許可 DDL ステートメント (例えば GRANT ステートメント) を生成します。このパラメーターは、**-t** および **-tw** パラメーターがいずれも指定されていない場合には、無視されます。サポートされている許可には、以下のものが含まれます。

- 列: UPDATE、REFERENCES
- 索引: CONTROL
- ストアード・プロシージャー: EXECUTE
- 表: ALTER、SELECT、INSERT、DELETE、UPDATE、INDEX、REFERENCE、CONTROL
- 表スペース: USE
- ユーザー定義関数 (UDF): EXECUTE
- ユーザー定義メソッド: EXECUTE
- ビュー: SELECT、INSERT、DELETE、UPDATE、CONTROL

-xd オブジェクト作成時に SYSIBM によって許可が付与されたオブジェクト用の許可 DDL ステートメントを含む、許可 DDL ステートメントを生成します。 システム・カタログ表とカタログ・ビューに対しては、許可 DDL は生成されません。

注: **-t** または **-tw** パラメーターのいずれかと共に使用した場合、このパラメーターは従属オブジェクトのための許可 DDL ステートメントを生成しません。親オブジェクトと従属オブジェクトに対して許可 DDL ステートメントを生成する場合は、**-xddep** パラメーターを使用してください。

-xddep オブジェクト作成時に SYSIBM によって権限を付与されたオブジェクトの許可 DDL ステートメントを含めて、親オブジェクトおよび従属オブジェクトに対して、すべての許可 DDL ステートメントを生成します。このパラメーターは、**-t** および **-tw** パラメーターがいずれも指定されていない場合には、無視されます。

-f 照会オプティマイザーに影響を与える構成パラメーターおよびレジストリー変数を抽出します。

-td *delimiter*

db2look コマンドによって生成される SQL ステートメントのステートメント区切り文字を指定します。デフォルトの区切り文字はセミコロン (;) です。抽出されたオブジェクトにはトリガーまたは SQL ルーチンが含まれる可能性があるので、**-e** パラメーターを指定する場合にはこのパラメーターを使用してください。

-noview

CREATE VIEW DDL ステートメントを抽出しないことを指定します。

-i *userid*

リモート・システムにログオンするために **db2look** コマンドが使用するユーザー ID を指定します。このパラメーターと **-w** パラメーターを指定すると、**db2look** コマンドをリモート・システム上のデータベースに対して実行することができます。ローカルとリモートのデータベースが、同じ DB2 バージョンを使用する必要があります。

-w *password*

リモート・システムにログオンするために **db2look** コマンドが使用するパスワードを指定します。このパラメーターと **-i** パラメーターを指定すると、**db2look** コマンドをリモート・システム上のデータベースに対して実行することができます。ローカルとリモートのデータベースが、同じ DB2 バージョンを使用する必要があります。

-wlm WLM 固有の DDL 出力を生成します。この出力は、以下のものに関する CREATE および ALTER ステートメントの生成に使用できます。

- ヒストグラム
- サービス・クラス
- しきい値
- WLM イベント・モニター
- ワークロード
- 作業アクション・セット

- 作業クラス・セット

-wrap ルーチン、トリガー、ビュー、PL/SQL パッケージの DDL ステートメントの難読化バージョンを生成します。

-wrapper *Wname*
指定したラッパーに適用するフェデレーテッド・オブジェクト用の DDL ステートメントを生成します。生成される可能性のあるフェデレーテッド DDL ステートメントには、以下のものがあります。

- CREATE FUNCTION ... AS TEMPLATE
- CREATE FUNCTION MAPPING
- CREATE INDEX SPECIFICATION
- CREATE NICKNAME
- CREATE SERVER
- CREATE TYPE MAPPING
- CREATE USER MAPPING
- CREATE WRAPPER
- GRANT (ニックネーム、サーバー、索引に対する特権)

ラッパー名を指定しない場合や、複数指定した場合には、エラーが戻されます。

-server *Sname*
指定したサーバーに適用するフェデレーテッド・オブジェクト用の DDL ステートメントを生成します。生成される可能性のあるフェデレーテッド DDL ステートメントには、以下のものがあります。

- CREATE FUNCTION ... AS TEMPLATE
- CREATE FUNCTION MAPPING
- CREATE INDEX SPECIFICATION
- CREATE NICKNAME
- CREATE SERVER
- CREATE TYPE MAPPING
- CREATE USER MAPPING
- CREATE WRAPPER
- GRANT (ニックネーム、サーバー、索引に対する特権)

サーバー名を指定しない場合や、複数指定した場合には、エラーが戻されます。

-nofed フェデレーテッド DDL ステートメントが生成されないことを指定します。このパラメーターを指定した場合、**-wrapper** および **-server** パラメーターは無視されます。

-fedonly
フェデレーテッド DDL ステートメントのみ生成されることを指定します。

-mod 各モジュール、および各モジュールに定義されているすべてのオブジェクト用に DDL ステートメントを生成します。

-xs XML スキーマと DTD をターゲット・データベースに登録するために必要

なすべてのファイルをエクスポートし、それらを登録するための該当するコマンドを生成します。エクスポートされる XSR オブジェクトのセットは、**-u**、**-z**、**-a** の各パラメーターによって制御されます。

-xdir *dirname*
XML 関連ファイルを指定されたパスにエクスポートします。このパラメーターを指定しない場合、XML 関連ファイルはすべて現行ディレクトリーにエクスポートされます。

-cor DDL ステートメントを CREATE OR REPLACE 節とともに生成します (ステートメントがその節をもともと含んでいるかどうかに関係なく)。

-noimplschema
暗黙的に作成されたスキーマに関する CREATE SCHEMA DDL ステートメントを生成しないように指定します。このパラメーターを指定する場合は **-e** パラメーターも指定する必要があります。

例

以下の例は、**db2look** コマンドの使用方法を示しています。

- データベース DEPARTMENT でユーザー walid によって作成されたオブジェクト用の DDL ステートメントを生成します。出力は、db2look.sql ファイルに送られます。

```
db2look -d department -u walid -e -o db2look.sql
```

- ianhe というスキーマ名を持ち、データベース DEPARTMENT でユーザー walid によって作成されたオブジェクト用の DDL ステートメントを生成します。出力は、db2look.sql ファイルに送られます。

```
db2look -d department -u walid -z ianhe -e -o db2look.sql
```

- UPDATE ステートメントを生成して、データベース DEPARTMENT でユーザー walid によって作成されたデータベース・オブジェクトの統計を複製します。出力は、db2look.sql ファイルに送られます。

```
db2look -d department -u walid -m -o db2look.sql
```

- ユーザー walid によって作成されたオブジェクト用の DDL ステートメントおよび UPDATE ステートメントの両方を生成して、同じユーザーによって作成されたデータベース・オブジェクトについての統計を複製します。出力は、db2look.sql ファイルに送られます。

```
db2look -d department -u walid -e -m -o db2look.sql
```

- データベース DEPARTMENT ですべてのユーザーによって作成されたオブジェクトの DDL ステートメントを生成します。出力は、db2look.sql ファイルに送られます。

```
db2look -d department -a -e -o db2look.sql
```

- すべてのユーザー定義のデータベース・パーティション・グループ、バッファー・プール、および表スペース用の DDL ステートメントを生成します。出力は、以下のようにして db2look.sql ファイルに書き込みます。

```
db2look -d department -l -o db2look.sql
```

- オプティマイザー関連のデータベースおよびデータベース・マネージャーの構成パラメーター用の UPDATE ステートメント、およびデータベース

DEPARTMENT にあるオプティマイザー関連のレジストリー変数用の **db2set** コマンドを生成します。出力は、db2look.sql ファイルに送られます。

```
db2look -d department -f -o db2look.sql
```

- オプティマイザー関連のレジストリー変数と、データベース DEPARTMENT のための以下のステートメントに対して **db2set** コマンドを生成します。
  - すべてのデータベース・オブジェクトのための DDL ステートメント
  - すべての表と索引についての統計を複製するための UPDATE ステートメント
  - GRANT 許可ステートメント
  - オプティマイザー関連のデータベースおよびデータベース・マネージャーの構成パラメーターに対する UPDATE ステートメント
  - オプティマイザー関連のレジストリー変数に対する **db2set** コマンド
  - すべてのユーザー定義のデータベース・パーティション・グループ、バッファー・プール、および表スペースに対する DDL ステートメント

  出力は、db2look.sql ファイルに送られます。

  ```
  db2look -d department -a -e -m -l -x -f -o db2look.sql
  ```
- オリジナルの作成者によって作成されたオブジェクトも含む、データベース DEPARTMENT 内のすべてのオブジェクトのすべての許可 DDL ステートメントを生成します。 (この場合には、オブジェクトの作成時に SYSIBM によって権限が付与されました。) 出力は、db2look.sql ファイルに送られます。

  ```
  db2look -d department -xd -o db2look.sql
  ```
- データベース DEPARTMENT ですべてのユーザーによって作成されたオブジェクトの DDL ステートメントを生成します。出力は、db2look.sql ファイルに送られます。

  ```
  db2look -d department -a -e -td % -o db2look.sql
  ```

  出力は CLP によって読み取ることができます。

  ```
  db2 -td% -f db2look.sql
  ```
- データベース DEPARTMENT 内のオブジェクト用に、 CREATE VIEW ステートメントを除く DDL ステートメントを生成します。出力は、db2look.sql ファイルに送られます。

  ```
  db2look -d department -e -noview -o db2look.sql
  ```
- 指定した表に関連するデータベース DEPARTMENT 内のオブジェクト用に、DDL ステートメントを生成します。出力は、db2look.sql ファイルに送られます。

  ```
  db2look -d department -e -t tab1 "¥"My TaBlE2¥"" -o db2look.sql
  ```
- フェデレーテッド・データベース FEDDEPART にすべてのオブジェクト (フェデレーテッドおよび非フェデレーテッド) 用の DDL ステートメントを生成します。フェデレーテッド DDL ステートメントでは、指定されたラッパー FEDWRAP に適用されるもののみが生成されます。出力が標準出力に書き込まれます。

  ```
  db2look -d feddepart -e -wrapper fedwrap
  ```
- 非フェデレーテッド DDL ステートメントのみを含むスクリプト・ファイルを生成します。以下のシステム・コマンドは、フェデレーテッド・データベース

FEDDEPART に対して実行でき、フェデレーテッドではないデータベースの実行時に検出されたような出力を生成するだけです。出力は、out.sql ファイルに送られます。

```
db2look -d feddepart -e -nofed -o out
```

- データベース DEPARTMENT の中でスキーマ名が walid であるオブジェクト用の DDL ステートメントを生成します。組み込み XML スキーマおよび DTD を登録するために必要なファイルは、現行ディレクトリーにエクスポートされます。出力は、db2look.sql ファイルに送られます。

  ```
  db2look -d department -z walid -e -xs -o db2look.sql
  ```

- データベース DEPARTMENT ですべてのユーザーによって作成されたオブジェクトの DDL ステートメントを生成します。組み込み XML スキーマおよび DTD を登録するために必要なファイルは、/home/ofer/ofer/ ディレクトリーにエクスポートされます。出力が標準出力に書き込まれます。

  ```
  db2look -d department -a -e -xs -xdir /home/ofer/ofer/
  ```

- データベース DEPARTMENT に対する WLM 固有の DDL ステートメントだけを生成します。

  ```
  db2look -d department -wlm
  ```

- データベース DEPARTMENT 内のすべてのオブジェクトに対する DDL ステートメントを生成します。

  ```
  db2look -d department -wlm -e -l
  ```

- スキーマ TABLES 内の親表 TAB1 と、TAB1 の従属ビュー (データベース DB1 内の VIEWS スキーマにあり VIEW1 と呼ばれる) の両方に対して DDL ステートメントを生成します。出力は、db2look.sql ファイルに送られます。

  ```
  db2look -d DB1 -t TABLES.TAB1 -e -o db2look.sql
  ```

- スキーマ TABLES 内の親表 TAB1 と、TAB1 の従属ビュー (データベース DB1 内の VIEWS スキーマにあり VIEW1 と呼ばれる) に対して DDL ステートメントおよび許可 DDL ステートメントを生成します。出力は、db2look.sql ファイルに送られます。

  ```
  db2look -d DB1 -t TABLES.TAB1 -e -xdep -o db2look.sql
  ```

- 模倣モードで TABLE1 表に RUNSTATS DDL ステートメントを生成します。出力は、db2look.sql ファイルに送られます。 このコマンドで使用可能な RUNSTATS 節すべてがサポートされているわけではありません。

  ```
  db2look -d DB1 -t TABLE1 -m  -e -o db2look.sql
  ```

使用上の注意

Windows オペレーティング・システムでは、**db2look** コマンドは DB2 コマンド・ウィンドウから発行する必要があります。

デフォルトでは、インスタンス所有者には **db2look** パッケージに対する EXECUTE 特権があります。その他のユーザーが **db2look** コマンドを実行する場合、インスタンス所有者は **db2look** パッケージに対する EXECUTE 特権を付与する必要があります。 **db2look** パッケージ名を判別するには、以下のようにして **db2bfd** コマンドを使用できます。

```
cd .../sqllib/bnd
db2bfd -b db2look.bnd
db2bfd -b db2lkfun.bnd
db2bfd -b db2lksp.bnd
```

フェデレーテッド・オブジェクトのための DDL ステートメントを作成するためには、データベース・マネージャー構成でフェデレーテッド・システムを使用可能に設定する必要があります。 **db2look** コマンドがスクリプト・ファイルを生成した後、そのスクリプトを実行する前に、**federated** 構成パラメーターを YES に設定する必要があります。以下の **db2look** コマンド・パラメーターがフェデレーテッド環境でサポートされます。

-ap
AUDIT USING ステートメントを生成します。

-e フェデレーテッド・オブジェクト用の DDL ステートメントを生成します。

-f フェデレーテッド関連情報をデータベース・マネージャー構成から抽出します。

-m ニックネームの統計を抽出します。

-x フェデレーテッド・オブジェクトに対する特権を付与する GRANT ステートメントを生成します。

-xd
フェデレーテッド・オブジェクトに対してシステムが付与する特権を追加する DDL ステートメントを生成します。

-wlm
WLM 固有の DDL ステートメントを生成します。

ニックネーム列とリモート表列が異なるデータ型である場合には、**db2look** コマンドは、ニックネーム列に対する ALTER COLUMN ステートメントを生成します。

出力スクリプトを変更して、CREATE USER MAPPING ステートメントのリモート・パスワードを追加する必要があります。

DB2 ファミリー・インスタンスをデータ・ソースとして定義するのに使用される、これらの CREATE SERVER ステートメントに、AUTHORIZATION および PASSWORD を追加して、**db2look** コマンド出力スクリプトを変更する必要があります。

-tw オプションの使用法は、次のとおりです。

- abc で始まる名前を持つ表に関連した、 DEPARTMENT データベース内のオブジェクトの DDL ステートメントを生成し、その出力を db2look.sql ファイルに送信するには、次のようにします。

  ```
  db2look -d department -e -tw abc% -o db2look.sql
  ```

- 名前の 2 番目の文字が d である表に関連した、 DEPARTMENT データベース内のオブジェクトの DDL ステートメントを生成し、その出力を db2look.sql ファイルに送信するには、次のようにします。

  ```
  db2look -d department -e -tw _d% -o db2look.sql
  ```

- **db2look** コマンドは、LIKE 述部を使用して、どの表名が *Tname* 引数に指定されたパターンに一致するかを評価します。 LIKE 述部を使用する以上、_ 文字または % 文字のいずれかが表名の一部である場合には、_ または % のすぐ前に円記

号 (¥) エスケープ文字を置かなければなりません。この場合、_ も % も、*Tname* 内でワイルドカード文字として使用することはできません。例えば、名前の最初でも最後でもない場所にパーセント (%) 記号を持つ表に関連した、DEPARTMENT データベース内のオブジェクトの DDL ステートメントを生成するには、次のようにします。

```
db2look -d department -e -tw string¥%string
```

- 大/小文字の区別がある名前、DBCS 名、複数語表名および複数語ビュー名は、円記号および二重引用符の両方で囲む必要があります。例えば、

```
¥"My TabLe¥"
```

  マルチバイト文字セット (MBCS) 名または 2 バイト文字セット (DBCS) 名が円記号と二重引用符の区切り文字で囲まれていない場合、小文字と同じバイトが含まれていると大文字に変換され、**db2look** は変換された名前を使用してデータベース・オブジェクトを探します。その結果、DDL ステートメントは抽出されません。

- **-tw** オプションは、**-x** オプション (GRANT 特権を生成する場合)、**-m** オプション (表統計および列統計を戻す場合)、および **-l** オプション (ユーザー定義表スペース、データベース・パーティション・グループ、およびバッファー・プールの DDL を生成する場合) とともに使用できます。 **-t** オプションを **-tw** オプションとともに指定すると、**-t** オプション (およびそれに関連した *Tname* 引数) が無視されます。
- フェデレーテッド・データ・ソース上、または DB2 Universal Database for z/OS and OS/390、DB2 for i、または DB2 Server for VSE & VM 上にある表 (およびそれらに関連したオブジェクト) の DDL を生成するために **-tw** オプションを使用することはできません。
- **-tw** オプションは、CLP でのみサポートされています。

パーティション・データベース環境のあるシステムで DDL ステートメントを生成しようとすると、非アクティブ・データベース・パーティション上にある表スペースに関しては、DDL ステートメントの代わりに警告メッセージが表示されます。すべての表スペースに対して正しい DDL ステートメントが作成されるようにするため、すべてのデータベース・パーティションをアクティブにする必要があります。

DB2 クライアントから、クライアントと同じかそれ以降のリリースのデータベースには **db2look** コマンドを発行できますが、クライアントからクライアントより前のリリースのデータベースにはこのコマンドは発行できません。例えば、**db2look** コマンドをバージョン 9.8 クライアントからバージョン 10.1 データベースに発行することはできますが、このコマンドをバージョン 10.1 クライアントからバージョン 9.8 データベースに発行することはできません。

db2look ユーティリティーを呼び出すと、**db2look** コマンドによって、非コミット・トランザクションを使用して作成されたオブジェクトに対して DDL ステートメントが生成されます。

タイプ配列のセキュリティー・ラベル・コンポーネントのための DDL ステートメントを抽出する場合、抽出されたステートメントが生成するコンポーネントの内部表記 (その配列内のエレメントのエンコード) が、**db2look** コマンドの発行対象のデータベース内のコンポーネントの内部表記と一致しないことがあります。この不一

致は、セキュリティー・ラベル・コンポーネントに 1 つ以上のエレメントを追加することによってコンポーネントを変更した場合に発生する可能性があります。そのような場合には、ある表から抽出されて **db2look** 出力から作成した別の表に移動したデータには、対応するセキュリティー・ラベル値がないので、新しい表の保護が危険にさらされる可能性があります。

関連情報:

ニックネーム列および索引の名前

マイグレーションのためのアプリケーションの変更

INGEST、インポート、およびロードの各ユーティリティーの比較

次の表に、INGEST、インポート、およびロードの各ユーティリティー間の重要な類似点および相違点を要約します。

表 17. サポートされる表タイプ

表タイプ	INGEST	ロード	インポート
デタッチされた表	サポートされていない	サポートされていない	サポートされていない
グローバル一時表	サポートされていない	サポートされていない	サポートされていない
マルチディメンション・クラスタリング (MDC) 表または挿入時クラスタリング (ITC) 表	サポートされている	サポートされている	サポートされている
ユーザー保守のマテリアライズ照会表 (MQT)	サポートされている	サポートされている	サポートされている
ニックネーム	サポートされている	サポートされていない	サポートされている
範囲がクラスター化された表 (RCT)	サポートされている	サポートされていない	サポートされている
範囲パーティション表	サポートされている	サポートされている	サポートされている
サマリー表	サポートされている	サポートされている	サポートされている
テンポラル表	サポートされている	サポートされている	サポートされている
型付き表	サポートされていない	サポートされていない	サポートされている
型なし表 (通常の表)	サポートされている	サポートされている	サポートされている
更新可能なビュー (型付きビューを除く)	サポートされている	サポートされていない	サポートされている

表 18. サポートされるデータ・タイプ

表タイプ	INGEST	ロード	インポート
数値: SMALLINT、INTEGER、 BIGINT、DECIMAL、 REAL、DOUBLE、 DECFLOAT	サポートされている	サポートされている	サポートされている

表 18. サポートされるデータ・タイプ (続き)

表タイプ	INGEST	ロード	インポート
文字: CHAR、VARCHAR、 NCHAR、NVARCHAR、および対応する FOR BIT DATA タイプ	サポートされている	サポートされている	サポートされている
グラフィック: GRAPHIC、VARGRAPHIC	サポートされている	サポートされている	サポートされている
ロング・タイプ: LONG VARCHAR、LONG VARGRAPHIC	サポートされている	サポートされている	サポートされている
日付/時刻: DATE、TIME、および TIMESTAMP(p) を含む TIMESTAMP	サポートされている	サポートされている	サポートされている
DB2SECURITYLABEL	サポートされている	サポートされている	サポートされている
ファイル内の LOB: BLOB、CLOB、 DBCLOB、NCLOB	サポートされていない	サポートされている	サポートされている
インライン LOB	サポートされていない	サポートされている	サポートされている
ファイル内の XML	サポートされていない	サポートされている	サポートされている
インライン XML	サポートされていない	サポートされている	サポートされている
特殊タイプ	サポートされている (サポートされる組み込みデータ・タイプに基づく場合)	サポートされている	サポートされている
構造化タイプ	サポートされていない	サポートされていない	サポートされている
参照タイプ	サポートされている	サポートされている	サポートされている

表 19. サポートされる入力ソース

入力タイプ	INGEST 再開可能性	ロード 再開可能性	インポート 再開可能性
カーソル	サポートされていない なし	サポートされている あり	サポートされていない なし
デバイス	サポートされていない なし	サポートされている あり	サポートされていない なし
ファイル	サポートされている あり	サポートされている あり	サポートされている あり
パイプ	サポートされている あり	サポートされている あり	サポートされていない なし

表 20. サポートされる入力フォーマット

表タイプ	INGEST	ロード	インポート
ASC (バイナリーを含む)	サポートされている	サポートされている	サポートされている
DB2 z/OS UNLOAD フォーマット	サポートされていない	サポートされていない	サポートされていない
DEL	サポートされている	サポートされている	サポートされている
IXF	サポートされていない	サポートされている	サポートされている

ロード・ユーティリティーとインポート・ユーティリティーを INGEST ユーティリティーと比較したときの重要な相違点が、他にもいくつかあります。

- INGEST ユーティリティーでは、入力レコードの中で、各列に対応するフィールドの間に追加フィールドを含めることができます。
- INGEST ユーティリティーは、更新、削除、およびマージをサポートします。
- INGEST ユーティリティーは、フィールド値を含んだ式から列の値を構成する機能をサポートします。
- INGEST ユーティリティーは、INGEST の実行中に他のアプリケーションがターゲット表を更新することを許可します。

ファイル形式とデータ・タイプ

エクスポート/インポート/ロード・ユーティリティーのファイル形式

ここでは、DB2 のエクスポート、インポート、およびロード・ユーティリティーによってサポートされている 4 種類のオペレーティング・システム・ファイル・フォーマットについて説明します。

DEL 区切り付き ASCII。さまざまなデータベース・マネージャーおよびファイル・マネージャーの間でのデータ交換に使用されます。このデータ保管の一般的なアプローチでは、列値を分離するために特殊な区切り文字が使用されます。

ASC 区切りなし ASCII。位置合わせされた列データをもつフラット・テキスト・ファイルを作成する他のアプリケーションからのデータのインポートまたはロードに使用されます。

PC/IXF

PC バージョンの IXF (統合交換フォーマット)。データベース・マネージャー内でのデータ交換のために望ましい方式です。 PC/IXF は、内部表の外部表記の入ったデータベース表の構造化された記述です。

CURSOR

SQL 照会に対して宣言されるカーソル。このファイル・タイプは、ロード・ユーティリティーによってのみサポートされます。

DEL または ASC データ・ファイル・フォーマットを使用する場合は、ファイルをインポートする前に、表 (その列名とデータ・タイプを含む) を定義してください。オペレーティング・システム・ファイルのフィールドのデータ・タイプは、データベース表内の対応するデータ・タイプに変換されます。インポート・ユーティリティーは、多少の非互換性問題があるデータを受け入れます。これには、埋め込みまたは切り捨ての可能性を伴って文字データをインポートする場合、および数値データをそれとは異なるタイプの数値フィールドにインポートする場合などがあります。

PC/IXF データ・ファイル・フォーマットを使用する場合、インポート操作の前に表が存在している必要はありません。ただし、ユーザー定義特殊タイプ (UDT) は、定義する必要があります。定義しなければ、未定義名エラー (SQL0204N) を受け取ることになります。同じように、PC/IXF データ・ファイル・フォーマットにエクスポートする場合は、UDT が出力ファイルに格納されます。

CURSOR ファイル・タイプを使用する際には、ロード操作を開始する前に、列名およびデータ・タイプなどで表を定義する必要があります。 SQL 照会の列タイプは、ターゲット表の対応する列タイプと互換性がなければなりません。ロード操作を開始する前に、指定されたカーソルをオープンする必要はありません。ロード・ユーティリティーは、カーソルが行のフェッチに使用されていたかどうかに関係なく、指定されたカーソルに関連する照会の結果全体を処理します。

プラットフォーム間のデータの移動 - ファイル形式に関する考慮事項

プラットフォームを超えてデータをエクスポート、インポート、またはロードする場合、互換性は重要です。以下の部分では、異なるオペレーティング・システム間でデータを移動する際の PC/IXF および区切り付き ASCII (DEL) ファイル・フォーマットの考慮事項について説明します。

PC/IXF ファイル・フォーマット

PC/IXF は、プラットフォーム間でデータを転送する際に望ましいファイル・フォーマットです。 PC/IXF を使用すると、ロード・ユーティリティーやインポート・ユーティリティーは、通常はマシンによって異なる数値データをマシンから独立した形で処理できます。例えば、Intel とその他のハードウェア体系とでは、数値データの保管および処理の方法が異なります。

DB2 ファミリー全製品で PC/IXF ファイルの互換性を保つために、エクスポート・ユーティリティーは Intel フォーマットの数値データによるファイルを作成し、インポート・ユーティリティーはこのフォーマットでデータを受け入れることを想定します。

ハードウェア・プラットフォームによっては、エクスポートおよびインポート操作中に、DB2 製品はバイト反転を使用して数値を Intel フォーマットから非 Intel フォーマットに変換します。

UNIX オペレーティング・システムをベースにした DB2 データベースの実装では、エクスポート中に複数パーツの PC/IXF ファイルを作成しません。しかし、DB2 の作成した複数パーツの PC/IXF ファイルをインポートすることはできます。このタイプのファイルをインポートする場合は、すべての部分を同じディレクトリーに入れる必要があります。そうしない場合エラーが戻されます。

DB2 エクスポート・ユーティリティーを使用して UNIX オペレーティング・システム上に作成された単一パーツの PC/IXF ファイルは、Windows 用の DB2 データベースでインポート可能です。

区切り付き ASCII (DEL) ファイル・フォーマット

DEL ファイルは、それらが作成されたオペレーティング・システムによって異なっています。相違点は次のとおりです。

- 行区切り文字
  - UNIX オペレーティング・システムのテキスト・ファイルでは、改行 (LF) 文字を使用します。
  - その他のオペレーティング・システムのテキスト・ファイルでは、復帰/改行 (CRLF) シーケンスを使用します。
- ファイル終了文字
  - UNIX オペレーティング・システムのテキスト・ファイルでは、ファイル終了文字を使用しません。
  - その他のオペレーティング・システムのテキスト・ファイルでは、ファイル終了文字 (X'1A') を使用します。

DEL エクスポート・ファイルはテキスト・ファイルなので、あるオペレーティング・システムから別のオペレーティング・システムに転送できます。テキスト・モードでファイルを転送する場合、ファイル転送プログラムを使用すると、オペレーティング・システムによる違いを処理できます。バイナリー・モードの場合、行区切り文字やファイル終了文字の変換は実行されません。

注: 文字データ・フィールドに行区切り文字が入っていれば、それらもまたファイル転送中に変換されます。そのような変換によって想定外のデータの変更が発生するため、プラットフォームを超えてデータを移動する場合は DEL エクスポート・ファイルを使わないことをお勧めします。その代わりに、PC/IXF ファイル・フォーマットを使用してください。

区切り付き ASCII (DEL) ファイル・フォーマット

区切り付き ASCII (DEL) ファイルは、行および列の区切り文字を使った順次 ASCII ファイルです。各 DEL ファイルは、行と列によって配列されたセル値から構成される ASCII 文字のストリームです。データ・ストリーム内の行は行区切り文字によって区切られ、それぞれの行の中の個々のセル値は列区切り文字によって区切られます。

以下の表は、インポートしたり、またはエクスポート・アクションの結果として生成したりできる DEL ファイルのフォーマットを示します。

```
DEL ファイル ::= 行 1 のデータ || 行区切り文字 ||
                 行 2 のデータ || 行区切り文字 ||
                 .
                 .
                 .
                 行 n のデータ || オプションの行区切り文字

行 i のデータ ::= セル値 (i,1) || 列区切り文字 ||
                  セル値 (i,2) || 列区切り文字 ||
                  .
```

```
                    .
                    .
                    セル値 (i,m)

行区切り文字 ::= ASCII 改行シーケンス[a]

列区切り文字 ::= デフォルト値 ASCII コンマ (,)[b]

セル値 (i,j) ::= 先行スペース
                 || 数値の ASCII 表記
                    (整数、10 進数、または浮動小数点数)
                 || 区切り文字ストリング
                 || 非区切り文字ストリング
                 || 後続スペース

非区切り文字ストリング ::= 行区切りまたは列区切り文字
                            以外の文字の集合

区切り文字ストリング ::=    文字ストリング区切り文字      ||
                               外字ストリング             ||
                               文字ストリング区切り文字    ||
                               後続ガーベッジ

後続ガーベッジ ::=    行区切り文字または列区切り文字以外の
                      文字の集合

文字ストリング区切り文字 ::= デフォルト値 ASCII 二重引用符記号
                              (")[c]

外字ストリング ::=          || NODOUBLEDEL 修飾子が指定されている場合、
                               文字ストリング区切り文字の
                               一部ではない場合、
                               文字の集合
                            || 文字ストリングが 2 つの連続した
                               文字ストリング区切り文字の一部ではない場合、
                               行区切り文字または文字ストリング区切り文字
                               以外の文字の集合

                            || 文字ストリング区切り文字が 2 つの連続した
                               文字ストリング区切り文字の一部ではなく、
                               かつ DELPRIORITYCHAR 修飾子が指定されて
                               いる場合、文字ストリング区切り文字以外の
                               文字の集合

ファイル終了 (EOF) 文字 ::= Hex '1A' (Windows オペレーティング・システムのみ)

数値の ASCII 表記 [d] ::= オプションの符号 '+' または '-'
    || 1 から 31 桁の 10 進数字に、オプションとして小数点をその前、その後、
       または数字と数字の間に入れたもの
    || オプションの指数

指数 ::= 文字 'E' または 'e'
    || オプションの符号 '+' または '-'
    || 1 から 3 桁の数字 (小数点なし)

10 進数字 ::= 文字 '0'、'1'、... '9' のいずれか

小数点 ::= デフォルト値 ASCII ピリオド (.)[e]
```

- [a] レコード区切り文字は、改行文字 (ASCII x0A) であると見なされます。 Windows オペレーティング・システムで生成されるデータでは、復帰/改行の 2 バイト標準 (0x0D0A) が使用できます。 EBCDIC コード・ページのデータでは、レコード区切り文字として EBCDIC LF 文字 (0x25) を使用します (EBCDIC データは、**LOAD** コマンドの code page ファイル・タイプ修飾子を使用してロードできます)。

- [b] 列区切り文字は、coldel ファイル・タイプ修飾子で指定できます。
- [c] 文字ストリング区切り文字は、chardel ファイル・タイプ修飾子で指定できます。

  注: 区切り文字のデフォルトの優先順位は、次のとおりです。

  1. レコード区切り文字
  2. 区切り文字
  3. 列区切り文字
- [d] 数値の ASCII 表記に指数が使用されている場合、それは FLOAT 定数です。小数点を使っているが指数は使っていない場合、それは DECIMAL 定数です。小数点も指数も使用されていない場合、それは INTEGER 定数です。
- [e] 小数点文字は、decpt ファイル・タイプ修飾子で指定できます。

エクスポート・ユーティリティーは、列データの中に埋め込まれたすべての文字ストリング区切り文字バイト (デフォルトは二重引用符または x22) を 2 つの文字ストリング区切り文字バイトに置き換えます (つまり、2 倍にします)。これは、インポート構文解析ルーチンが、列の開始または終了を定義する文字ストリング区切り文字バイトと、列データの中に埋め込まれた文字ストリング区切り文字バイトを区別できるようにするために行われます。エクスポート・ユーティリティー以外の何らかのアプリケーション用にエクスポートされた DEL ファイルを使用するときは注意を払い、この同じ文字ストリング区切り文字の倍増が、'FOR BIT' バイナリー列データの中で発生することに注意してください。

DEL のデータ・タイプの説明:

表 21. DEL ファイル・フォーマットで受け入れられるデータ・タイプのフォーマット

データ・タイプ	エクスポート・ユーティリティーによって作成されるファイルの形式	インポート・ユーティリティーにとって受け入れ可能な形式
BIGINT	-9,223,372,036,854,775,808 から 9,223,372,036,854,775,807 の範囲の INTEGER 定数。	-9,223,372,036,854,775,808 から 9,223,372,036,854,775,807 の範囲の数値の ASCII 表記。 10 進数および浮動小数点数は整数値に切り捨てられます。
BLOB、CLOB	区切り文字 (例えば二重引用符) によって囲まれた文字データ。	区切り文字ストリングまたは区切りなし文字ストリング。文字ストリングは、データベースの列値として使用されます。
BLOB_FILE、CLOB_FILE	各 BLOB/CLOB 列ごとに文字データが個別のファイルに保管され、そのファイル名は区切り文字によって囲まれます。	データが入っているファイルの区切り付き名前または区切りなし名前。

表 21. DEL ファイル・フォーマットで受け入れられるデータ・タイプのフォーマット (続き)

データ・タイプ	エクスポート・ユーティリティーによって作成されるファイルの形式	インポート・ユーティリティーにとって受け入れ可能な形式
CHAR	区切り文字 (例えば二重引用符) によって囲まれた文字データ。	区切り文字ストリングまたは区切りなし文字ストリング。文字ストリングは、必要な場合データベース列の幅に合わせて切り捨てられるか、またはスペース (X'20') が埋められます。
DATE	区切り文字なしの *yyyymmdd* (年、月、日)。例えば、19931029。 あるいは、DATESISO オプションを使用して、すべての日付値が ISO フォーマットでエクスポートされるように指定することもできます。	ターゲット・データベースの地域別コードに一致する ISO フォーマットの日付値の入った区切り付きまたは区切りなし文字ストリング、あるいは *yyyymmdd* の形式の区切りなし文字ストリング。
DBCLOB (DBCS のみ)	GRAPHIC データは、区切り付き文字ストリングとしてエクスポートされます。	偶数バイトの長さの区切り付きまたは区切りなし文字ストリング。文字ストリングは、データベースの列値として使用されます。
DBCLOB_FILE (DBCS のみ)	各 DBCLOB 列ごとに文字データが個別のファイルに保管され、そのファイル名は区切り文字によって囲まれます。	データが入っているファイルの区切り付き名前または区切りなし名前。
DB2SECURITYLABEL	列のデータは、引用符 (") で囲まれて「生の」データとしてエクスポートされます。この値をセキュリティー・ラベル・ストリング・フォーマットに変換するには、 SELECT ステートメントの中で SECLABEL_TO_CHAR スカラー関数を使用します。	デフォルトでは、データ・ファイル内の値は、そのセキュリティー・ラベルの内部表記を構成する実際のバイト数と見なされます。値は引用符 (") で囲まれているものと想定されます。
DECIMAL	エクスポートされるフィールドの精度および位取りによる DECIMAL 定数。decplusblank ファイル・タイプ修飾子を使用して、正の 10 進値の前に正符号 (+) ではなくブランク・スペースを付けるように指定することも可能です。	フィールドのインポート先のデータベース列の範囲からあふれない数値の ASCII 表記。入力値の小数部の列数が、データベース列で受け入れ可能なものより多い場合、余分な列は切り捨てられます。
FLOAT(long)	-10E307 から 10E307 の範囲の FLOAT 定数。	-10E307 から 10E307 の範囲の数値の ASCII 表記。

表 21. DEL ファイル・フォーマットで受け入れられるデータ・タイプのフォーマット (続き)

データ・タイプ	エクスポート・ユーティリティーによって作成されるファイルの形式	インポート・ユーティリティーにとって受け入れ可能な形式
GRAPHIC (DBCS のみ)	GRAPHIC データは、区切り付き文字ストリングとしてエクスポートされます。	偶数バイトの長さの区切り付きまたは区切りなし文字ストリング。文字ストリングは、必要な場合データベース列の幅に合わせて切り捨てられるか、または 2 バイト・スペース (例えば X'8140') が埋められます。
INTEGER	-2,147,483,648 から 2,147,483,647 の範囲の INTEGER 定数。	-2,147,483,648 から 2,147,483,647 の範囲の数値の ASCII 表記。 10 進数および浮動小数点数は整数値に切り捨てられます。
LONG VARCHAR	区切り文字 (例えば二重引用符) によって囲まれた文字データ。	区切り文字ストリングまたは区切りなし文字ストリング。文字ストリングは、データベースの列値として使用されます。
LONG VARGRAPHIC (DBCS のみ)	GRAPHIC データは、区切り付き文字ストリングとしてエクスポートされます。	偶数バイトの長さの区切り付きまたは区切りなし文字ストリング。文字ストリングは、データベースの列値として使用されます。
SMALLINT	-32,768 から 32,767 の範囲の INTEGER 定数。	-32,768 から 32,767 の範囲の数値の ASCII 表記。 10 進数および浮動小数点数は整数値に切り捨てられます。
TIME	*hh.mm.ss* (時、分、秒)。区切り文字によって囲まれた ISO フォーマットの時刻値。例えば「09.39.43」。	ターゲット・データベースの地域別コードに合致するフォーマットの時刻値を使った区切り付きまたは区切りなし文字ストリング。
TIMESTAMP	*yyyy-mm-dd-hh.mm.ss.nnnnnn* (年、月、日、時、分、秒、マイクロ秒)。区切り文字によって囲まれた日時を表す文字ストリング。	データベースでの保管用に受け入れ可能なタイム・スタンプ値を使った区切り付きまたは区切りなし文字ストリング。
VARCHAR	区切り文字 (例えば二重引用符) によって囲まれた文字データ。	区切り文字ストリングまたは区切りなし文字ストリング。文字ストリングは、必要な場合データベース列の最大幅に合わせて切り捨てられます。

表 21. DEL ファイル・フォーマットで受け入れられるデータ・タイプのフォーマット (続き)

データ・タイプ	エクスポート・ユーティリティーによって作成されるファイルの形式	インポート・ユーティリティーにとって受け入れ可能な形式
VARGRAPHIC (DBCS のみ)	GRAPHIC データは、区切り付き文字ストリングとしてエクスポートされます。	偶数バイトの長さの区切り付きまたは区切りなし文字ストリング。文字ストリングは、必要な場合データベース列の最大幅に合わせて切り捨てられます。

DEL ファイル例:

DEL ファイルの例を下記に示します。各行は改行文字シーケンスで終わります (Windows オペレーティング・システムでは、各行は復帰/改行文字シーケンスで終わります)。

```
"Smith, Bob",4973,15.46
"Jones, Bill",12345,16.34
"Williams, Sam",452,193.78
```

次の例は、区切り文字なし文字ストリングの使用方法を示しています。文字データ中にコンマが使われているため、列区切り文字はセミコロンに変更されています。

```
Smith, Bob;4973;15.46
Jones, Bill;12345;16.34
Williams, Sam;452;193.78
```

注:

1. スペース (X'20') は有効な区切り文字ではありません。
2. セル値の最初の文字の前または最後の文字の後にあるスペースは、インポート時に破棄されます。セル値の途中にあるスペースは破棄されません。
3. ピリオド (.) は、タイム・スタンプ値のピリオドと競合するため、有効な文字ストリング区切り文字ではありません。
4. DBCS のみ (GRAPHIC)、混合 DBCS、および EUC の場合、区切り文字の範囲は x00 から x3F に制限されます。
5. EBCDIC コード・ページで指定された DEL データの場合、区切り文字は DBCS のシフトイン文字およびシフトアウト文字と同じであってはなりません。
6. Windows オペレーティング・システムの場合、区切り文字の中にない最初のファイル終了文字 (X'1A') がファイル終わりを示します。それ以降のデータはインポートされません。
7. NULL 値は、通常はセル値があるはずの場所にセル値がないことによって、あるいはスペースのストリングによって示されます。
8. 一部の製品では文字フィールドが 254 または 255 バイトに制限されるため、エクスポート・ユーティリティーは、最大長が 254 バイトより長い文字タイプの列がエクスポート用に選択されるたびに警告メッセージを生成します。インポート・ユーティリティーは、最も長い LONG VARCHAR および LONG VARGRAPHIC 列と同じ長さのフィールドを受け入れます。

データ移動時の区切り文字に関する考慮事項:

区切り文字が使用されている ASCII (DEL) ファイルを移動する場合は、区切り文字の認識方法にかかわる問題のために、移動するデータをうっかり変更してしまわないようにする必要があります。このようなエラーを回避するために、DB2 には、いくつかの制約が設定されており、そのためのファイル・タイプ修飾子が用意されています。

区切り文字に関する制約事項

特定の区切り文字が、移動するデータの一部として処理される事態を回避するために、いくつかの制約が設定されています。第 1 に、区切り文字は、相互に排他的です。第 2 に、区切り文字として、2 進ゼロ、改行文字、復帰文字、ブランク・スペースを使用することはできません。さらに、デフォルトの小数点 (.) をストリング区切り文字として使用することはできません。最後に、DBCS 環境では、ストリング区切り文字としてパイプ (|) がサポートされていません。

ASCII 系のコード・ページと EBCDIC 系のコード・ページでは、以下の文字の指定方法が異なります。

- EBCDIC MBCS データ・ファイルでは、シフトイン (0x0F) 文字とシフトアウト (0x0E) 文字を区切り文字として使用できません。
- MBCS コード・ページ、EUC コード・ページ、DBCS コード・ページでは、0x40 より大きな文字を区切り文字として使用できません。ただし、EBCDIC MBCS データのデフォルトの小数点 (0x4b) は例外です。
- ASCII コード・ページまたは EBCDIC MBCS コード・ページのデータ・ファイルのデフォルトの区切り文字は、以下のとおりです。
  - ストリング区切り文字: "(0x22、二重引用符)
  - 列区切り文字: ,(0x2c、コンマ)
- EBCDIC SBCS コード・ページのデータ・ファイルのデフォルトの区切り文字は、以下のとおりです。
  - ストリング区切り文字: "(0x7F、二重引用符)
  - 列区切り文字: ,(0x6B、コンマ)
- ASCII データ・ファイルのデフォルトの小数点は、0x2e (ピリオド) です。
- EBCDIC データ・ファイルのデフォルトの小数点は、0x4B (ピリオド) です。
- サーバーのコード・ページとクライアントのコード・ページが異なる場合は、デフォルト以外の区切り文字の 16 進表記を指定することをお勧めします。例えば、次のようにします。

```
db2 load from ... modified by chardel0x0C coldelX1e ...
```

データ移動時の区切り文字に関する問題

二重のストリング区切り文字

DEL ファイルの文字ベース・フィールドの場合は、フィールド内にストリング区切り文字が見つかると、そのすべての出現箇所が二重のストリング区切り文字で表記される、というのがデフォルトの動作です。例えば、ストリング区切り文字が二重引用符の場合に、`I am 6" tall.` というテキストをエクスポートすると、DEL ファイルの出力テキストは、`"I am 6"" tall."` に

なります。その逆に、DEL ファイルの入力テキストが "What a ""nice"" day!" になっていると、そのテキストは、What a "nice" day! という形でインポートされます。

nodoubledel
インポート/エクスポート/ロード・ユーティリティーで二重のストリング区切り文字の動作を無効にするには、nodoubledel ファイル・タイプ修飾子を指定します。ただし、二重のストリング区切り文字の動作が存在しているのは構文解析エラーを回避するためである、という点は忘れないようにする必要があります。エクスポートで nodoubledel を使用するときは、ストリング区切り文字が文字フィールドにあっても、そのストリング区切り文字は二重になりません。インポートとロードで nodoubledel を使用すると、二重のストリング区切り文字は、ストリング区切り文字のリテラル・インスタンスとして解釈されなくなります。

nochardel
エクスポートで nochardel ファイル・タイプ修飾子を使用すると、文字フィールドがストリング区切り文字で囲まれなくなります。インポートとロードで nochardel を使用すると、ストリング区切り文字は、特殊文字としてではなく実際のデータとして解釈されます。

chardel
その他のファイル・タイプ修飾子を使用して、デフォルトの区切り文字とデータの混同を手動で回避することも可能です。chardel ファイル・タイプ修飾子では、二重引用符 (デフォルト) の代わりに使用する文字ストリング区切り文字として、x という 1 文字を指定します。

coldel
同じように、列区切り文字としてデフォルトのコンマを使用したくない場合は、coldel を使用できます。この修飾子では、列データ区切り文字として、x という 1 文字を指定します。

delprioritychar
DEL ファイルの移動に伴うもう 1 つの問題は、区切り文字の正しい優先順位を維持することです。区切り文字のデフォルトの優先順位は、行、文字 (桁)、列です。ただし、中には、文字 (桁)、行、列という優先順位に依存するアプリケーションもあります。例えば、デフォルトの優先順位を使用する次のような DEL データ・ファイルがあるとします。

```
"Vincent <row delimiter> is a manager",<row delimiter>
```

このデータは、Vincent と is a manager という 2 つの行として解釈されます。行区切り文字 (<row delimiter>) の方がストリング区切り文字 (") よりも優先順位が高いからです。一方、delprioritychar を使用すると、ストリング区切り文字 (") の方が行区切り文字 (<row delimiter>) よりも優先順位が高くなり、同じ DEL ファイルでも、この場合は Vincent is a manager という 1 行として (正しく) 解釈されます。

区切りなし ASCII (ASC) ファイル形式

区切りなし ASCII フォーマット (インポートおよびロード・ユーティリティーでは ASC と言う) には、固定長 ASC と可変長 ASC の 2 つの種類があります。固定長

の ASC の場合、すべてのレコードが固定長になります。可変長の ASC の場合、レコードは行区切り文字 (常に改行) で区切られます。区切りなし ASCII における *区切りなし* という言葉は、列の値が区切り文字によって区切られないということを意味します。

ASC データのインポートまたはロード時に reclen ファイル・タイプ修飾子を指定すると、データ・ファイルは固定長の ASC になります。これを指定しないと、データ・ファイルは可変長の ASC になります。

区切りなし ASCII フォーマットは、ワード・プロセッサーを含め、縦欄フォーマットのデータを使用する ASCII 製品とのデータ交換に使用することができます。各 ASC ファイルは、行と列によって配列されたデータ値から構成される ASCII 文字のストリームです。データ・ストリーム内の行は行区切り文字によって区切られます。行内の各列は、開始/終了ロケーションの対 (**IMPORT** パラメーターで指定される) によって定義されます。それぞれの対は、バイト・ロケーションとして指定された行内のロケーションを表します。行内の最初の位置はバイト位置 1 です。それぞれの対の最初のエレメントは列の開始バイト、2 番目のエレメントは列の終了バイトです。列が互いに重なり合うことも可能です。 1 つの ASC ファイル内の各行の列定義はすべて同じです。

ASC ファイルの定義は、次のとおりです。

```
ASC ファイル ::= 行 1 のデータ || 行区切り文字 ||
                 行 2 のデータ || 行区切り文字 ||
                 .
                 .
                 .
                 行 n のデータ

行 i のデータ ::= ASCII 文字 || 行区切り文字

行区切り文字 ::= ASCII 改行シーケンス[a]
```

- [a] レコード区切り文字は、改行文字 (ASCII x0A) であると見なされます。Windows オペレーティング・システムで生成されるデータでは、復帰/改行の 2 バイト標準 (0x0D0A) が使用できます。 EBCDIC コード・ページのデータでは、レコード区切り文字として EBCDIC LF 文字 (0x25) を使用します (EBCDIC データは、**LOAD** コマンドの codepage ファイル・タイプ修飾子を使用してロードできます)。レコード区切り文字がデータのフィールドの一部として解釈されることはありません。

ASC のデータ・タイプの説明:

表 22. *ASC* ファイル・フォーマットで受け入れられるデータ・タイプのフォーマット

データ・タイプ	インポート・ユーティリティーにとって受け入れ可能な形式
BIGINT	すべての数値タイプ (SMALLINT、 INTEGER、 BIGINT、 DECIMAL、または FLOAT) の定数が受け入れられます。 -9,223,372,036,854,775,808 から 9,223,372,036,854,775,807 の範囲外の値があれば、その特定の値に関してはリジェクトされます。 10 進数は整数値に切り捨てられます。コンマ、ピリオド、またはコロンは小数点であると見なされます。 3 桁ごとの区切り文字は使用できません。 開始ロケーションと終了ロケーションは、幅が 50 バイト以下のフィールドになるように指定してください。整数、10 進数、および浮動小数点数の小数部は 31 桁以下です。浮動小数点数の指数は 3 桁以下です。
BLOB/CLOB	文字のストリング。文字ストリングは、必要な場合ターゲット列の最大長に合わせて右側が切り捨てられます。 ASC のブランク *切り捨て* オプションが有効な場合は、元のストリングまたは切り捨てられたストリングから後書きブランクが取り除かれます。
BLOB_FILE、 CLOB_FILE、 DBCLOB_FILE (DBCS のみ)	データが入っているファイルの区切り付き名前または区切りなし名前。
CHAR	文字のストリング。文字ストリングは、必要な場合ターゲット列の幅に合わせて切り捨てられるか、またはスペースが埋められます。
DATE	ターゲット・データベースの地域別コードに合致するフォーマットの日付値を表す文字ストリング。 開始ロケーションと終了ロケーションは、日付の外部表記の範囲内のフィールド幅になるように指定してください。
DBCLOB (DBCS のみ)	偶数バイトのストリング。奇数バイトのストリングは無効であり受け入れられません。有効なストリングは、必要な場合ターゲット列の最大長に合わせて右側が切り捨てられます。
DECIMAL	すべての数値タイプ (SMALLINT、 INTEGER、 BIGINT、 DECIMAL、または FLOAT) の定数が受け入れられます。インポート先のデータベース列の範囲外の値があれば、その特定の値に関してはリジェクトされます。入力値の小数部の桁が、データベース列の位取りより多い場合、余分な桁は切り捨てられます。コンマ、ピリオド、またはコロンは小数点であると見なされます。 3 桁ごとの区切り文字は使用できません。 開始ロケーションと終了ロケーションは、幅が 50 バイト以下のフィールドになるように指定してください。整数、10 進数、および浮動小数点数の小数部は 31 桁以下です。浮動小数点数の指数は 3 桁以下です。

表 22. ASC ファイル・フォーマットで受け入れられるデータ・タイプのフォーマット (続き)

データ・タイプ	インポート・ユーティリティーにとって受け入れ可能な形式
FLOAT(long)	すべての数値タイプ (SMALLINT、 INTEGER、 BIGINT、 DECIMAL、または FLOAT) の定数が受け入れられます。すべての値が有効です。コンマ、ピリオド、またはコロンは小数点であると見なされます。大文字または小文字の E は、FLOAT 定数の指数の始まりとして受け入れられます。 開始ロケーションと終了ロケーションは、幅が 50 バイト以下のフィールドになるように指定してください。整数、10 進数、および浮動小数点数の小数部は 31 桁以下です。浮動小数点数の指数は 3 桁以下です。
GRAPHIC (DBCS のみ)	偶数バイトのストリング。奇数バイトのストリングは無効であり受け入れられません。有効なストリングは、必要な場合ターゲット列の最大長に合わせて右側が切り捨てられるか、または 2 バイト・スペース (0x8140) が埋められます。
INTEGER	すべての数値タイプ (SMALLINT、 INTEGER、 BIGINT、 DECIMAL、または FLOAT) の定数が受け入れられます。 -2,147,483,648 から 2,147,483,647 の範囲外の整数があれば、それに関してはリジェクトされます。 10 進数は整数値に切り捨てられます。コンマ、ピリオド、またはコロンは小数点であると見なされます。 3 桁ごとの区切り文字は使用できません。 開始ロケーションと終了ロケーションは、幅が 50 バイト以下のフィールドになるように指定してください。整数、10 進数、および浮動小数点数の小数部は 31 桁以下です。浮動小数点数の指数は 3 桁以下です。
LONG VARCHAR	文字のストリング。文字ストリングは、必要な場合ターゲット列の最大長に合わせて右側が切り捨てられます。 ASC のブランク *切り捨て* オプションが有効な場合は、元のストリングまたは切り捨てられたストリングから後書きブランクが取り除かれます。
LONG VARGRAPHIC (DBCS のみ)	偶数バイトのストリング。奇数バイトのストリングは無効であり受け入れられません。有効なストリングは、必要な場合ターゲット列の最大長に合わせて右側が切り捨てられます。
SMALLINT	すべての数値タイプ (SMALLINT、 INTEGER、 BIGINT、 DECIMAL、または FLOAT) の定数が受け入れられます。 -32,768 から 32,767 の範囲外の値があれば、その特定の値に関してはリジェクトされます。 10 進数は整数値に切り捨てられます。コンマ、ピリオド、またはコロンは小数点であると見なされます。 3 桁ごとの区切り文字は使用できません。 開始ロケーションと終了ロケーションは、幅が 50 バイト以下のフィールドになるように指定してください。整数、10 進数、および浮動小数点数の小数部は 31 桁以下です。浮動小数点数の指数は 3 桁以下です。
TIME	ターゲット・データベースの地域別コードに合致するフォーマットの時間値を表す文字ストリング。 開始ロケーションと終了ロケーションは、時間の外部表記の範囲内のフィールド幅になるように指定してください。

表 22. *ASC* ファイル・フォーマットで受け入れられるデータ・タイプのフォーマット *(続き)*

データ・タイプ	インポート・ユーティリティーにとって受け入れ可能な形式
TIMESTAMP	データベースでの保管用に受け入れ可能なタイム・スタンプ値を表す文字ストリング。 開始ロケーションと終了ロケーションは、タイム・スタンプの外部表記の範囲内のフィールド幅になるように指定してください。
VARCHAR	文字のストリング。文字ストリングは、必要な場合ターゲット列の最大長に合わせて右側が切り捨てられます。 ASC のブランク切り捨て オプションが有効な場合は、元のストリングまたは切り捨てられたストリングから後書きブランクが取り除かれます。
VARGRAPHIC (DBCS のみ)	偶数バイトのストリング。奇数バイトのストリングは無効であり受け入れられません。有効なストリングは、必要な場合ターゲット列の最大長に合わせて右側が切り捨てられます。

ASC ファイル例:

ASC ファイルの例を下記に示します。各行は改行文字シーケンスで終わります (Windows オペレーティング・システムでは、各行は復帰/改行文字シーケンスで終わります)。

```
Smith, Bob       4973     15.46
Jones, Suzanne   12345    16.34
Williams, Sam    452123  193.78
```

注:

1. ASC ファイルには、列名が入っていないものと見なされます。
2. 文字ストリングは、区切り文字によって囲まれません。 ASC ファイルの列のデータ・タイプは、データベース表のターゲット列のデータ・タイプによって決まります。
3. 次の場合、NULL 可能データベース列に NULL がインポートされます。
   - ブランクのフィールドのターゲットが数値、DATE、TIME、または TIMESTAMP タイプのデータベース列である場合
   - 開始/終了ロケーションの対のないフィールドが指定されている場合
   - 長さが 0 になる開始/終了ロケーションの対が指定されている場合
   - ある行のデータが短すぎて、ターゲット列にとって有効な値が入っていない場合
   - NULL INDICATORS ロード・オプションが使用されていて、NULL 標識列に N (またはユーザーによって指定されたその他の値) が検出された場合
4. NULL 可能でないターゲット列に数値、DATE、TIME、または TIMESTAMP 列にブランクのフィールドをインポートしようとすると、その行はリジェクトされます。
5. 入力データにターゲット列との互換性がなく、その列が NULL 可能でない場合は、エラーが検出された場所に応じて NULL がインポートされるか、または行がリジェクトされます。列が NULL 可能でない場合は、行がリジェクトされます。互換性がないことを示すメッセージがメッセージ・ファイルに書き込まれます。

PC バージョンの IXF ファイル形式

PC バージョンの IXF (PC/IXF) ファイル形式は、データベース・マネージャーに適応させた統合交換フォーマット (IXF) データ交換アーキテクチャーです。 IXF アーキテクチャーは、リレーショナル・データベースの構造とデータの交換を可能にするために特に設計されたものです。 PC/IXF アーキテクチャーを使用すれば、データベース・マネージャーは、データベースのエクスポート時に受信側の製品の要件や特性を予測する必要がなくなります。同じように、PC/IXF ファイルをインポートする側の製品で必要なことも、PC/IXF アーキテクチャーを理解するだけになります。ファイルをエクスポートした製品の特性は影響しなくなります。 PC/IXF ファイル・アーキテクチャーは、エクスポート側とインポート側の両方のデータベース・システムからの独立性を保ちます。

IXF アーキテクチャーは、特定のリレーショナル・データベース製品によってサポートされていない一部のタイプを含め、リレーショナル・データ・タイプの豊富なセットをサポートする汎用リレーショナル・データベース交換フォーマットです。PC/IXF ファイル・フォーマットではこの柔軟性が保たれます。例えば、PC/IXF アーキテクチャーでは、1 バイト文字セット (SBCS) と 2 バイト文字セット (DBCS) の両方のデータ・タイプがサポートされます。すべてのインプリメンテーションですべての PC/IXF データ・タイプがサポートされるわけではありませんが、制限付きのインプリメンテーションでも、インポート操作において、サポートされないデータ・タイプの検出と後処理が実行されます。

一般に PC/IXF ファイルは、中断なしの一連の可変長レコードから構成されます。ファイルには、次のレコードが入れられます。順序はここに示す順序です。

- レコード・タイプ H の 1 つのヘッダー・レコード
- レコード・タイプ T の 1 つの表レコード
- レコード・タイプ C の複数の列記述子レコード (表の列ごとに 1 つのレコード)
- レコード・タイプ D の複数のデータ・レコード (表の各行が 1 つまたは複数の D レコードによって表現されます)。

PC/IXF ファイルにおいて、H レコードの後であればどこにでもアプリケーション (A) レコードを入れることができます。 PC/IXF ファイルにおいてそれらのレコードは、PC/IXF フォーマットで定義されていない追加のデータをアプリケーションが PC/IXF ファイルに組み込むことができるようになっています。 PC/IXF ファイルを読み取るプログラムに、A レコード内のアプリケーション ID によって暗黙指定されるデータ・フォーマットと内容に関する特別の知識がない場合、そのレコードは無視されます。

PC/IXF ファイル内のレコードは、いずれもレコード長標識で始まります。これは、PC/IXF レコードのうちこのレコード長標識より後の部分の長さをバイト単位で指定する整数値の、6 バイトの右寄せ文字表記です (合計レコード・サイズ - 6 バイト)。 PC/IXF ファイルを読むプログラムは、これらのレコード長を使用することにより、現行レコードの終わりと次のレコードの始まりを検出します。 H、T、および C レコードは、それらに定義されているすべてのフィールドを入れるのに十分な大きさでなければならず、当然のこととしてそれらのレコード長フィールドは、それらの実際の長さと一致していなければなりません。しかし、これらのいずれかのレコードの終わりに余分なデータ (例えば新しい フィールド) が追加された場合、

PC/IXF ファイルを読む従来のプログラムは余分のデータを無視し、警告メッセージしか生成しません。ただし、PC/IXF ファイルに書き込むプログラムの場合は、すべての定義済みフィールドを入れるのにちょうど必要な長さの H、T、および C レコードを書き込まなければなりません。

PC/IXF ファイルに LOB ロケーション指定子 (LLS) 列が入っている場合には、それぞれの LLS 列ごとに D レコードがなければなりません。 D レコードはエクスポート・ユーティリティーによって自動的に作成されますが、 PC/IXF ファイルを生成するためにサード・パーティーのツールを使用している場合には、これらを手動で作成する必要があります。さらに、NULL 値のある列を含め、表内の各 LOB 列ごとに LLS が必要です。 LOB 列が NULL である場合には、 NULL LOB を表す LLS を作成する必要があります。

各 XML 列の D レコード項目には、2 バイトのリトル・エンディアン (XML データ指定子 (XDS) の長さを示す) と、それに続く XDS そのものが入ります。

例えば、以下のような XDS があるとします。

```
XDS FIL="a.xml" OFF="1000" LEN="100" SCH="RENATA.SCHEMA" />
```

これは、次の D レコードのバイトによって表されます。

```
0x3D 0x00 XDS FIL="a.xml" OFF="1000" LEN="100" SCH="RENATA.SCHEMA" />
```

PC/IXF ファイルのレコードは、文字データの入ったフィールドで構成されます。インポートおよびエクスポート・ユーティリティーは、ターゲット・データベースの CPGID を使用してこの文字データを解釈します。ただし、2 つの例外があります。

- A レコードの IXFADATA フィールド。

  IXFADATA フィールド内に入れられる文字データのコード・ページ環境は、特定の A レコードを作成および処理するアプリケーションによって設定されます。つまり、環境は実装ごとに違います。
- D レコードの IXFDCOLS フィールド。

  IXFDCOLS フィールド内に入れられる文字データのコード・ページ環境は、特定の列とそのデータを定義する C レコードに入っている情報によって決まります。

H、T、および C レコード内の数値フィールド、そして D および A レコードの接頭部内の数値フィールドは、整数値の 1 バイト文字表記を右寄せして先行ゼロまたはブランクを埋め込んだものでなければなりません。値 0 は、ブランクではなく、少なくとも 1 つの (右寄せされた) 0 の文字によって示されなければなりません。長さがデータ・タイプによって暗黙指定される数値フィールド (例えば IXFCLENG) が使用されない場合、それにはブランクを埋め込む必要があります。そのような数値フィールドは、次のとおりです。

```
IXFHRECL, IXFTRECL, IXFCRECL, IXFDRECL, IXFARECL,
IXFHHCNT, IXFHSBCP, IXFHDBCP, IXFTCCNT, IXFTNAML,
IXFCLENG, IXFCDRID, IXFCPOSN, IXFCNAML, IXFCTYPE,
IXFCSBCP, IXFCDBCP, IXFCNDIM, IXFCDSIZ, IXFDRID
```

注: データベース・マネージャーの PC/IXF ファイル形式は、System/370 と同じではありません。

PC/IXF レコード・タイプ:

PC/IXF の基本レコード・タイプには、次の 5 種類があります。

- ヘッダー
- 表
- 列記述子
- データ
- アプリケーション

これに、DB2 が使用する、以下の 7 つのアプリケーション・サブタイプが加わります。

- 索引
- 階層
- 副表
- 継続
- 終了
- ID
- DB2 SQLCA

PC/IXF の各レコード・タイプは一連のフィールドとして定義されます。それらのフィールドは必須であり、示されている順序になっていなければなりません。

ヘッダー・レコード

フィールド名	長さ	タイプ	備考
IXFHRECL	06-BYTE	CHARACTER	レコード長
IXFHRECT	01-BYTE	CHARACTER	レコード・タイプ = 'H'
IXFHID	03-BYTE	CHARACTER	IXF ID
IXFHVERS	04-BYTE	CHARACTER	IXF のバージョン
IXFHPROD	12-BYTE	CHARACTER	製品
IXFHDATE	08-BYTE	CHARACTER	作成日付
IXFHTIME	06-BYTE	CHARACTER	作成時刻
IXFHHCNT	05-BYTE	CHARACTER	ヘッダー・レコードのカウント
IXFHSBCP	05-BYTE	CHARACTER	単一バイト・コード・ページ
IXFHDBCP	05-BYTE	CHARACTER	2 バイト・コード・ページ
IXFHFIL1	02-BYTE	CHARACTER	予約済み

ヘッダー・レコードには、以下のフィールドが入っています。

IXFHRECL
レコード長標識。 PC/IXF レコードのうちこのレコード長標識より後の部分の長さをバイト単位で指定する整数値の 6 バイトの右寄せ文字表記 (合計レコード・サイズ - 6 バイト)。 H レコードは、そのすべての定義済みフィールドを入れるのに十分な長さでなければなりません。

IXFHRECT
IXF レコード・タイプ (このレコードの場合、H にセットされる)。

IXFHID
ファイル・フォーマット ID (このファイルの場合、IXF にセットされる)。

IXFHVERS
ファイルの作成時に使用された PC/IXF フォーマット・レベル (0002 にセットされる)。

IXFHPROD

ファイルを作成しているプログラムが、それ自体を識別するために使用できるフィールド。このフィールドにデータが入っている場合、その最初の 6 バイトはファイルを作成した製品を識別するために使用され、最後の 6 バイトはその製品のバージョンまたはリリースを示すために使用されます。データベース・マネージャーは、データベース・マネージャー固有データの存在を通知するためにこのフィールドを使用します。

IXFHDATE

ファイルの作成日付 (*yyyymmdd* の形式)。

IXFHTIME

ファイルの作成時刻 (*hhmmss* の形式)。このフィールドはオプションであり、ブランクのままにしておくことができます。

IXFHHCNT

このファイルのうち最初のデータ・レコードより前にある H、T、および C レコードの数。 A レコードは、このカウントに入りません。

IXFHSBCP

SBCS CPGID または '00000' の 1 バイト文字表記の入った 1 バイト・コード・ページ・フィールド。

エクスポート・ユーティリティーは、このフィールドを、エクスポートされるデータベース表の SBCS CPGID と等しい値にセットします。例えば、表の SBCS CPGID が 850 の場合、このフィールドの内容は '00850' です。

IXFHDBCP

DBCS CPGID または '00000' の 1 バイト文字表記の入った 2 バイト・コード・ページ・フィールド。

エクスポート・ユーティリティーは、このフィールドを、エクスポートされるデータベース表の DBCS CPGID と等しい値にセットします。例えば、表の DBCS CPGID が 301 の場合、このフィールドの内容は '00301' です。

IXFHFIL1

ホスト IXF ファイルの予約フィールドに一致させるために、2 つのブランクにセットされるスペア・フィールド。

表レコード

フィールド名	長さ	タイプ	備考
IXFTRECL	006-BYTE	CHARACTER	レコード長
IXFTRECT	001-BYTE	CHARACTER	レコード・タイプ = 'T'
IXFTNAML	003-BYTE	CHARACTER	名前の長さ
IXFTNAME	256-BYTE	CHARACTER	データの名前
IXFTQULL	003-BYTE	CHARACTER	修飾子の長さ
IXFTQUAL	256-BYTE	CHARACTER	修飾子
IXFTSRC	012-BYTE	CHARACTER	データ・ソース
IXFTDATA	001-BYTE	CHARACTER	データ規則 = 'C'
IXFTFORM	001-BYTE	CHARACTER	データ・フォーマット = 'M'
IXFTMFRM	005-BYTE	CHARACTER	マシン形式 = 'PC'
IXFTLOC	001-BYTE	CHARACTER	データ・ロケーション = 'I'
IXFTCCNT	005-BYTE	CHARACTER	'C' レコード・カウント
IXFTFIL1	002-BYTE	CHARACTER	予約済み
IXFTDESC	030-BYTE	CHARACTER	データの記述
IXFTPKNM	257-BYTE	CHARACTER	主キー名

```
IXFTDSPC        257-BYTE   CHARACTER   予約済み
IXFTISPC        257-BYTE   CHARACTER   予約済み
IXFTLSPC        257-BYTE   CHARACTER   予約済み
```

表レコードには、以下のフィールドが入っています。

IXFTRECL
レコード長標識。 PC/IXF レコードのうちこのレコード長標識より後の部分の長さをバイト単位で指定する整数値の 6 バイトの右寄せ文字表記 (合計レコード・サイズ - 6 バイト)。 T レコードは、そのすべての定義済みフィールドを入れるのに十分な長さでなければなりません。

IXFTRECT
IXF レコード・タイプ (このレコードの場合、T にセットされる)。

IXFTNAML
IXFTNAME フィールド内の表名の長さ (バイト単位)。

IXFTNAME
表の名前。各ファイルごとに 1 つの表だけがある場合、これは単なる情報フィールドです。データベース・マネージャーは、データのインポート時にこのフィールドを使用しません。 PC/IXF ファイルへの書き込み時にデータベース・マネージャーは、DOS ファイル名 (場合によってはパス情報) をこのフィールドに書き込みます。

IXFTQULL
IXFTQUAL フィールド内の表名修飾子の長さ (バイト単位)。

IXFTQUAL
リレーショナル・システム内で表の作成者を識別する表名修飾子。これは単なる情報フィールドです。ファイルに書き込むプログラムにこのフィールドに書き込むデータがない場合、推奨される充てん値はブランクです。ファイルを読み取るプログラムでは、このフィールドを印刷または表示したり、情報フィールドに保管したりできますが、このフィールドの内容に基づく演算は信頼性がありません。

IXFTSRC
データのオリジナル・ソースを指示するために使用されます。これは単なる情報フィールドです。ファイルに書き込むプログラムにこのフィールドに書き込むデータがない場合、推奨される充てん値はブランクです。ファイルを読み取るプログラムでは、このフィールドを印刷または表示したり、情報フィールドに保管したりできますが、このフィールドの内容に基づく演算は信頼性がありません。

IXFTDATA
データの記述に使用される規則。インポートまたはエクスポートの場合、このフィールドは C にセットしなければなりません。これは、個々の列属性が次の列記述子 (C) レコードで記述されており、データが PC/IXF 規則に従っていることを示します。

IXFTFORM
数値データの保管に使用される規則。このフィールドは、M にセットしなければなりません。これは、データ (D) レコード内の数値データが IXFTMFRM フィールドによって指定されるマシン (内部) フォーマットで保管されていることを示します。

IXFTMFRM

PC/IXF ファイル内のマシン・データのフォーマット。データベース・マネージャーは、このフィールドが PC*bbb* にセットされている場合にのみ、ファイルの読み取りまたは書き込みを実行します。 *b* はブランクを表し、PC は、PC/IXF ファイルのデータが IBM PC マシン・フォーマットになっていることを指定します。

IXFTLOC

データのロケーション。データベース・マネージャーは、値として I のみサポートします。これは、データがこのファイルの内部にあることを意味します。

IXFTCCNT

この表内の C レコードの数。これは、整数値の右寄せ文字表記です。

IXFTFIL1

ホスト IXF ファイルの予約フィールドに一致させるために、2 つのブランクにセットされるスペア・フィールド。

IXFTDESC

表に関する記述データ。これは単なる情報フィールドです。ファイルに書き込むプログラムにこのフィールドに書き込むデータがない場合、推奨される充てん値はブランクです。ファイルを読み取るプログラムでは、このフィールドを印刷または表示したり、情報フィールドに保管したりできますが、このフィールドの内容に基づく演算は信頼性がありません。列がデフォルトによって NULL になっておらず、表名がワークステーションのデータベースからのものである場合、このフィールドには NOT NULL WITH DEFAULT が入ります。

IXFTPKNM

表で定義されている主キーの名前 (ある場合)。この名前は NULL 文字で終了するストリングとして格納されます。

IXFTDSPC

このフィールドは将来の利用のために予約済み。

IXFTISPC

このフィールドは将来の利用のために予約済み。

IXFTLSPC

このフィールドは将来の利用のために予約済み。

列記述子レコード

フィールド名	長さ	タイプ	備考
IXFCRECL	006-BYTE	CHARACTER	レコード長
IXFCRECT	001-BYTE	CHARACTER	レコード・タイプ = 'C'
IXFCNAML	003-BYTE	CHARACTER	列名長
IXFCNAME	256-BYTE	CHARACTER	列名
IXFCNULL	001-BYTE	CHARACTER	列が NULL 可能
IXFCDEF	001-BYTE	CHARACTER	列にデフォルトがある
IXFCSLCT	001-BYTE	CHARACTER	列選択フラグ
IXFCKPOS	002-BYTE	CHARACTER	主キーでの位置
IXFCCLAS	001-BYTE	CHARACTER	データ・クラス
IXFCTYPE	003-BYTE	CHARACTER	データ・タイプ
IXFCSBCP	005-BYTE	CHARACTER	単一バイト・コード・ページ
IXFCDBCP	005-BYTE	CHARACTER	2 バイト・コード・ページ
IXFCLENG	005-BYTE	CHARACTER	列データ長
IXFCDRID	003-BYTE	CHARACTER	'D' レコード ID

```
IXFCPOSN      006-BYTE   CHARACTER   列位置
IXFCDESC      030-BYTE   CHARACTER   列記述
IXFCLOBL      020-BYTE   CHARACTER   LOB 列の長さ
IXFCUDTL      003-BYTE   CHARACTER   UDT 名の長さ
IXFCUDTN      256-BYTE   CHARACTER   UDT 名
IXFCDEFL      003-BYTE   CHARACTER   デフォルト値の長さ
IXFCDEFV      254-BYTE   CHARACTER   デフォルト値
IXFCREF       001-BYTE   CHARACTER   参照タイプ
IXFCNDIM      002-BYTE   CHARACTER   次元数
IXFCDSIZ      varying    CHARACTER   各次元のサイズ
```

列記述子レコードには、以下のフィールドが入っています。

IXFCRECL
レコード長標識。 PC/IXF レコードのうちこのレコード長標識より後の部分の長さをバイト単位で指定する整数値の 6 バイトの右寄せ文字表記 (合計レコード・サイズ - 6 バイト)。 C レコードは、そのすべての定義済みフィールドを入れるのに十分な長さでなければなりません。

IXFCRECT
IXF レコード・タイプ (このレコードの場合、C にセットされる)。

IXFCNAML
IXFCNAME フィールド内の列名の長さ (バイト単位)。

IXFCNAME
列の名前。

IXFCNULL
この列で NULL が可能かどうかを指定します。有効な設定は、Y または N です。

IXFCDEF
このフィールドのデフォルト値が定義されているかどうかを指定します。有効な設定は、Y または N です。

IXFCSLCT
データ中の列のサブセットの選択を可能にすることを目的としたフィールドで、現在は廃止されています。 PC/IXF ファイルへの書き込みを実行するプログラムでは、このフィールドに常に Y を保管しなければなりません。 PC/IXF ファイルを読むプログラムでは、このフィールドを無視しなければなりません。

IXFCKPOS
主キーの一部としての列の位置。有効値は 01 から 16 ですが、主キーの一部でない列の場合は N です。

IXFCCLAS
IXFCTYPE フィールドで使用されるデータ・タイプのクラス。データベース・マネージャーは、リレーショナル・タイプ (R) のみサポートします。

IXFCTYPE
列のデータ・タイプ。

IXFCSBCP
SBCS CPGID の 1 バイト文字表記。このフィールドは、この列の D レコードの IXFDCOLS フィールドに入れられる 1 バイト文字データの CPGID を指定します。

このフィールドのセマンティクスは、列のデータ・タイプ (IXFCTYPE フィールドで指定される) によって違います。

- 文字ストリング列の場合、このフィールドには、通常、H レコードの IXFHSBCP フィールドの値と等しいゼロ以外の値が入っていなければなりません。ただし、その他の値も許可されます。この値がゼロの場合、列はビット・ストリング・データが入ると解釈されます。
- 数値列の場合、このフィールドには意味がありません。このフィールドは、エクスポート・ユーティリティーでは 0 にセットされ、インポート・ユーティリティーでは無視されます。
- 日付列または時刻列の場合、このフィールドには意味がありません。このフィールドは、エクスポート・ユーティリティーでは IXFHSBCP フィールドの値にセットされ、インポート・ユーティリティーでは無視されます。
- GRAPHIC 列の場合、このフィールドは 0 でなければなりません。

IXFCDBCP

DBCS CPGID の 1 バイト文字表記。このフィールドは、この列の D レコードの IXFDCOLS フィールドに入れられる 2 バイト文字データの CPGID を指定します。

このフィールドのセマンティクスは、列のデータ・タイプ (IXFCTYPE フィールドで指定される) によって違います。

- 文字ストリング列の場合、このフィールドには、0 か、または H レコードの IXFHDBCP フィールドの値と等しい値が入っていなければなりません。ただし、その他の値も許可されます。 IXFCSBCP フィールドの値が 0 の場合、このフィールドの値は 0 でなければなりません。
- 数値列の場合、このフィールドには意味がありません。このフィールドは、エクスポート・ユーティリティーでは 0 にセットされ、インポート・ユーティリティーでは無視されます。
- 日付列または時刻列の場合、このフィールドには意味がありません。このフィールドは、エクスポート・ユーティリティーでは 0 にセットされ、インポート・ユーティリティーでは無視されます。
- GRAPHIC 列の場合、このフィールドの値は IXFHDBCP フィールドの値と同じでなければなりません。

IXFCLENG

記述されている列のサイズに関する情報を提供します。データ・タイプによっては、このフィールドは使用されず、ブランクを入れる必要があります。他のいくつかのデータ・タイプの場合、このフィールドには、列の長さを指定する整数の右寄せ文字表記が入れられます。また、さらに別のいくつかのデータ・タイプの場合、このフィールドは 2 つのサブフィールド (精度を表す 3 バイトと位取りを表す 2 バイト) に分割されます。これらのサブフィールドはいずれも整数の右寄せ文字表記です。バージョン 9.7 から、タイム・スタンプ・データ・タイプの場合に、このフィールドにはタイム・スタンプの精度を指定する整数の右寄せ文字表記が入るようになりました。

IXFCDRID

D レコード ID。このフィールドの内容は、整数値の右寄せ文字表記です。PC/IXF ファイルでは、各行のデータを入れるために複数の D レコードが

使用されることがあります。このフィールドは、あるデータ行を表現する D レコードのうち、どの D レコードに列のデータが入っているかを指定します。値 1 (例えば 001) は、列のデータがデータ行の最初の D レコードに入っていることを示します。最初の C レコードの IXFCDRID 値は 1 でなければなりません。それ以降のすべての C レコードの IXFCDRID 値は、その直前の C レコードと等しい値か、または 1 大きい値でなければなりません。

IXFCPOSN
このフィールドの値は、表のあるデータ行を表現する D レコードの 1 つの中で、列のデータを見つけるのに使用されます。これは、D レコードの IXFDCOLS フィールド内でのこの列のデータの開始位置です。列が NULL 可能なら IXFCPOSN は NULL 標識を指し、それ以外の場合にはデータ自体を指します。列に可変長データがある場合、データ自体が現行長標識で始まります。 D レコードの IXFDCOLS フィールド内の最初のバイトに対応する IXFCPOSN 値は 1 です (0 ではありません)。列が新しい D レコードに入っている場合、IXFCPOSN 値は 1 になります。それ以外の場合、IXFCPOSN 値はデータ値が重なり合わない範囲で列ごとに増加します。

IXFCDESC
列に関する記述情報。これは単なる情報フィールドです。ファイルに書き込むプログラムにこのフィールドに書き込むデータがない場合、推奨される充てん値はブランクです。ファイルを読み取るプログラムでは、このフィールドを印刷または表示したり、情報フィールドに保管したりできますが、このフィールドの内容に基づく演算は信頼性がありません。

IXFCLOBL
この列内で定義される long または LOB の長さ (バイト単位)。この列が long または LOB でない場合、このフィールドの値は 000 になります。

IXFCUDTL
IXFCUDTN フィールド内のユーザー定義タイプ (UDT) 名の長さ (バイト単位)。この列のタイプが UDT でない場合、このフィールドの値は 000 になります。

IXFCUDTN
この列のデータ・タイプとして使用されるユーザー定義タイプの名前。

IXFCDEFL
IXFCDEFV フィールド内のデフォルト値の長さ (バイト単位)。この列にデフォルト値がない場合、このフィールドの値は 000 になります。

IXFCDEFV
この列にデフォルト値が定義されている場合に、それを指定します。

IXFCREF
列が階層の一部を成している場合、このフィールドは、その列がデータ列 (D) または参照列 (R) のどちらであるかを指定します。

IXFCNDIM
列の次元の数。配列は、このバージョンの PC/IXF ではサポートされていません。したがって、このフィールドの内容は整数値 0 の文字表記でなければなりません。

IXFCDSIZ

各次元のサイズまたは範囲。このフィールドの長さは、1 次元当たり 5 バイトです。配列はサポートされない (次元の数は 0 でなければならない) ため、このフィールドは長さ 0 であり、実際には存在しません。

データ・レコード

フィールド名	長さ	タイプ	備考
IXFDRECL	06-BYTE	CHARACTER	レコード長
IXFDRECT	01-BYTE	CHARACTER	レコード・タイプ = 'D'
IXFDRID	03-BYTE	CHARACTER	'D' レコード ID
IXFDFIL1	04-BYTE	CHARACTER	予約済み
IXFDCOLS	varying	variable	縦欄データ

データ・レコードには、以下のフィールドが入っています。

IXFDRECL

レコード長標識。 PC/IXF レコードのうちこのレコード長標識より後の部分の長さをバイト単位で指定する整数値の 6 バイトの右寄せ文字表記 (合計レコード・サイズ - 6 バイト)。各 D レコードは、レコードに保管される最後のデータ列の現行オカレンスのすべての有効なデータを入れるのに十分な長さでなければなりません。

IXFDRECT

IXF レコード・タイプ (このレコードの場合、D にセットされる)。これは、このレコードに表のデータ値が入っていることを示します。

IXFDRID

レコード ID。これは、あるデータ行を表現する一連の D レコードの中で特定の D レコードを識別します。あるデータ行の最初の D レコードの場合、このフィールドの値は 1 になります。第 2 の D レコードの場合、このフィールドの値は 2 になり、以下同様です。各データ行の中には、C レコードの中で指定されているすべての D レコード ID が実際に存在していなければなりません。

IXFDFIL1

ホスト IXF ファイルの中で、予約フィールドに対応するブランク 4 個に設定されるスペア・フィールドで、使用される可能性があるシフトアウト文字のプレースホルダーとなるもの。

IXFDCOLS

縦欄データ用の領域。データ・レコード (D レコード) のデータ域は、1 つまたは複数の列項目で構成されます。その D レコードと同じ D レコード ID の列記述子レコードごとに、1 つの列項目があります。 D レコードにおける列項目の開始位置は、C レコードの IXFCPOSN 値によって指示されます。

列項目データのフォーマットは、列が NULL 可能であるかどうかによって異なります。

- 列が NULL 可能である場合 (IXFCNULL フィールドが Y にセットされている場合)、列項目データには NULL 標識が組み込まれます。列が NULL でない場合は、標識の後にデータ・タイプ固有の情報 (実際のデー

タベース値を含む) が続きます。 NULL 標識は、NULL でない場合は x'0000' にセットされ、NULL の場合は x'FFFF' にセットされる 2 バイトの値です。

- 列が NULL 可能でない場合、列項目データの内容はデータ・タイプ固有の情報 (実際のデータベース値を含む) だけです。

可変長データ・タイプの場合のデータ・タイプ固有の情報には、現行長標識が組み込まれます。現行長標識は、IXFTMFRM フィールドによって指定される形式の 2 バイトの整数です。

D レコードのデータ域の長さは、32,771 バイトを超えてはなりません。

アプリケーション・レコード

フィールド名	長さ	タイプ	備考
IXFARECL	06-BYTE	CHARACTER	レコード長
IXFARECT	01-BYTE	CHARACTER	レコード・タイプ = 'A'
IXFAPPID	12-BYTE	CHARACTER	アプリケーション ID
IXFADATA	varying	variable	アプリケーション固有のデータ

アプリケーション・レコードには、以下のフィールドが入っています。

IXFARECL
レコード長標識。 PC/IXF レコードのうちこのレコード長標識より後の部分の長さをバイト単位で指定する整数値の 6 バイトの右寄せ文字表記 (合計レコード・サイズ - 6 バイト)。各 A レコードは、少なくとも IXFAPPID フィールドの全体を入れるのに十分な長さでなければなりません。

IXFARECT
IXF レコード・タイプ (このレコードの場合、A にセットされる)。これがアプリケーション・レコードであることを示します。アプリケーション ID によって暗黙指定されるデータの内容とフォーマットに関する特別の知識がないプログラムでは、このレコードは無視されます。

IXFAPPID
アプリケーション ID。これは、A レコードを作成したアプリケーションを識別します。データベース・マネージャーによって作成された PC/IXF ファイルの A レコードの場合、このフィールドの最初の 6 文字を、データベース・マネージャーを識別する定数に設定して、最後の 6 文字を、データベース・マネージャーまたは A レコードを作成している別のアプリケーションのリリースまたはバージョンを識別する定数に設定することができます。

IXFADATA
このフィールドには、アプリケーションに依存する補足データが入れられます。その形式と内容は、A レコードを作成するプログラムと、A レコードを処理する他のアプリケーションによってのみ認識されます。

DB2 索引レコード

フィールド名	長さ	タイプ	備考
IXFARECL	006-BYTE	CHARACTER	レコード長
IXFARECT	001-BYTE	CHARACTER	レコード・タイプ = 'A'
IXFAPPID	012-BYTE	CHARACTER	application identifier = 'DB2 02.00'
IXFAITYP	001-BYTE	CHARACTER	application specific data type = 'I'
IXFADATE	008-BYTE	CHARACTER	'H' レコードから書き込まれた日付
IXFATIME	006-BYTE	CHARACTER	'H' レコードから書き込まれた時刻
IXFANDXL	002-BYTE	SHORT INT	索引名の長さ

```
IXFANDXN      256-BYTE   CHARACTER   索引名
IXFANCL       002-BYTE   SHORT INT   索引作成者名の長さ
IXFANCN       256-BYTE   CHARACTER   索引作成者名
IXFATABL      002-BYTE   SHORT INT   表名の長さ
IXFATABN      256-BYTE   CHARACTER   表名
IXFATCL       002-BYTE   SHORT INT   表作成者名の長さ
IXFATCN       256-BYTE   CHARACTER   表作成者名
IXFAUNIQ      001-BYTE   CHARACTER   ユニーク規則
IXFACCNT      002-BYTE   SHORT INT   column count
IXFAREVS      001-BYTE   CHARACTER   逆スキャン・フラグの許可
IXFAIDXT      001-BYTE   CHARACTER   type of index
IXFAPCTF      002-BYTE   CHARACTER   空き PCT の量
IXFAPCTU      002-BYTE   CHARACTER   最小使用 PCT の量
IXFAEXTI      001-BYTE   CHARACTER   予約済み
IXFACNML      006-BYTE   SHORT INT   length of name of the columns
IXFACOLN      varying    CHARACTER   索引内の列の名前
```

ユーザー定義索引ごとに、このタイプのレコードが 1 つずつ指定されます。このレコードは、表のすべての C レコードの後に置かれます。 DB2 索引レコードには、以下のフィールドが入っています。

IXFARECL
レコード長標識。 PC/IXF レコードのうちこのレコード長標識より後の部分の長さをバイト単位で指定する整数値の 6 バイトの右寄せ文字表記 (合計レコード・サイズ - 6 バイト)。各 A レコードは、少なくとも IXFAPPID フィールドの全体を入れるのに十分な長さでなければなりません。

IXFARECT
IXF レコード・タイプ (このレコードの場合、A にセットされる)。これがアプリケーション・レコードであることを示します。アプリケーション ID によって暗黙指定されるデータの内容とフォーマットに関する特別の知識がないプログラムでは、このレコードは無視されます。

IXFAPPID
アプリケーション ID。これは、A レコードを作成したアプリケーションとして DB2 を識別します。

IXFAITYP
これがサブタイプ "I" の DB2 アプリケーション・レコードであることを指定します。

IXFADATE
ファイルの作成日付 (*yyyymmdd* の形式)。このフィールドの値は、IXFHDATE と同じでなければなりません。

IXFATIME
ファイルの作成時刻 (*hhmmss* の形式)。このフィールドの値は、IXFHTIME と同じでなければなりません。

IXFANDXL
IXFANDXN フィールド内の索引名の長さ (バイト単位)。

IXFANDXN
索引の名前。

IXFANCL
IXFANCN フィールド内の索引作成者名の長さ (バイト単位)。

IXFANCN
索引作成者の名前。

IXFATABL
IXFATABN フィールド内の表名の長さ (バイト単位)。

IXFATABN
表の名前。

IXFATCL
IXFATCN フィールド内の表作成者名の長さ (バイト単位)。

IXFATCN
表作成者の名前。

IXFAUNIQ
索引のタイプを指定します。有効値は、P (主キー)、U (ユニーク索引)、および D (非ユニーク索引) です。

IXFACCNT
索引定義内の列の数を指定します。

IXFAREVS
この索引で逆スキャンを行えるかどうかを指定します。有効値は、 Y (逆スキャン可) と N (逆スキャン不可) です。

IXFAIDXT
索引のタイプを指定します。有効な値は R (通常の索引) および C (クラスター索引) です。

IXFAPCTF
空き状態にしておく索引ページのパーセントを指定します。有効値は -1 から 99 です。 -1 またはゼロの値を指定すると、システム・デフォルト値が使用されます。

IXFAPCTU
2 つの索引ページを組み合わせる場合に、あらかじめ空きになっている必要のある索引ページの最小パーセントを指定します。有効値は 00 から 99 の範囲です。

IXFAEXTI
将来の使用のために予約されています。

IXFACNML
IXFACOLN フィールド内の列名の長さ (バイト単位)。

IXFACOLN
この索引の一部を成す列の名前。有効値は *+name -name...* の形式です。 + は列の昇順ソートを指定し、 - は列の降順ソートを指定します。

DB2 階層レコード

フィールド名	長さ	タイプ	備考
IXFARECL	006-BYTE	CHARACTER	レコード長
IXFARECT	001-BYTE	CHARACTER	レコード・タイプ = 'A'
IXFAPPID	012-BYTE	CHARACTER	application identifier = 'DB2 02.00'
IXFAXTYP	001-BYTE	CHARACTER	application specific data type = 'X'
IXFADATE	008-BYTE	CHARACTER	'H' レコードから書き込まれた日付
IXFATIME	006-BYTE	CHARACTER	'H' レコードから書き込まれた時刻
IXFAYCNT	010-BYTE	CHARACTER	この階層の 'Y' レコード・カウント
IXFAYSTR	010-BYTE	CHARACTER	この階層の開始列

階層を記述する際、このタイプのレコードが 1 つ使用されます。すべての副表レコード (以下のリストを参照) は階層レコードの直後に置かれなければならず、階層レコードは表のすべての C レコードの後に置かれます。DB2 階層レコードには、以下のフィールドが入っています。

IXFARECL
レコード長標識。 PC/IXF レコードのうちこのレコード長標識より後の部分の長さをバイト単位で指定する整数値の 6 バイトの右寄せ文字表記 (合計レコード・サイズ - 6 バイト)。各 A レコードは、少なくとも IXFAPPID フィールドの全体を入れるのに十分な長さでなければなりません。

IXFARECT
IXF レコード・タイプ (このレコードの場合、A にセットされる)。これがアプリケーション・レコードであることを示します。アプリケーション ID によって暗黙指定されるデータの内容とフォーマットに関する特別の知識がないプログラムでは、このレコードは無視されます。

IXFAPPID
アプリケーション ID。これは、A レコードを作成したアプリケーションとして DB2 を識別します。

IXFAXTYP
これがサブタイプ "X" の DB2 アプリケーション・レコードであることを指定します。

IXFADATE
ファイルの作成日付 (*yyyymmdd* の形式)。このフィールドの値は、IXFHDATE と同じでなければなりません。

IXFATIME
ファイルの作成時刻 (*hhmmss* の形式)。このフィールドの値は、IXFHTIME と同じでなければなりません。

IXFAYCNT
この階層レコードの後の副表レコードの予想数を指定します。

IXFAYSTR
エクスポート・データの先頭における副表レコードの索引を指定します。階層のエクスポートがルート以外の副表から開始された場合、その副表のすべての親表がエクスポートされます。このフィールドには、IXF ファイル内のこの副表の位置も格納されます。最初の X レコードは、ゼロの索引をもつ列を表します。

DB2 副表レコード

フィールド名	長さ	タイプ	備考
IXFARECL	006-BYTE	CHARACTER	レコード長
IXFARECT	001-BYTE	CHARACTER	レコード・タイプ = 'A'
IXFAPPID	012-BYTE	CHARACTER	application identifier = 'DB2 02.00'
IXFAYTYP	001-BYTE	CHARACTER	application specific data type = 'Y'
IXFADATE	008-BYTE	CHARACTER	'H' レコードから書き込まれた日付
IXFATIME	006-BYTE	CHARACTER	'H' レコードから書き込まれた時刻
IXFASCHL	003-BYTE	CHARACTER	タイプ・スキーマ名の長さ
IXFASCHN	256-BYTE	CHARACTER	タイプ・スキーマ名
IXFATYPL	003-BYTE	CHARACTER	タイプ名の長さ
IXFATYPN	256-BYTE	CHARACTER	タイプ名
IXFATABL	003-BYTE	CHARACTER	表名の長さ
IXFATABN	256-BYTE	CHARACTER	表名
IXFAPNDX	010-BYTE	CHARACTER	親表の副表索引

```
IXFASNDX       005-BYTE   CHARACTER   現行の表の開始列索引
IXFAENDX       005-BYTE   CHARACTER   現行の表の終了列索引
```

階層の一部として副表を記述する際、このタイプのレコードが 1 つ使用されます。階層に属するすべての副表レコードは、対応する階層レコードの直後に、一緒に格納されている必要があります。副表は 1 つ以上の列で構成され、おのおのの列は列レコード内で記述されます。副表内の各列は、連続した C レコード・セット内で記述しなければなりません。 DB2 副表レコードには、以下のフィールドが入っています。

IXFARECL
レコード長標識。 PC/IXF レコードのうちこのレコード長標識より後の部分の長さをバイト単位で指定する整数値の 6 バイトの右寄せ文字表記 (合計レコード・サイズ - 6 バイト)。各 A レコードは、少なくとも IXFAPPID フィールドの全体を入れるのに十分な長さでなければなりません。

IXFARECT
IXF レコード・タイプ (このレコードの場合、A にセットされる)。これがアプリケーション・レコードであることを示します。アプリケーション ID によって暗黙指定されるデータの内容とフォーマットに関する特別の知識がないプログラムでは、このレコードは無視されます。

IXFAPPID
アプリケーション ID。これは、A レコードを作成したアプリケーションとして DB2 を識別します。

IXFAYTYP
これがサブタイプ "Y" の DB2 アプリケーション・レコードであることを指定します。

IXFADATE
ファイルの作成日付 (*yyyymmdd* の形式)。このフィールドの値は、IXFHDATE と同じでなければなりません。

IXFATIME
ファイルの作成時刻 (*hhmmss* の形式)。このフィールドの値は、IXFHTIME と同じでなければなりません。

IXFASCHL
IXFASCHN フィールド内の副表スキーマ名の長さ (バイト単位)。

IXFASCHN
副表スキーマの名前。

IXFATYPL
IXFATYPN フィールド内の副表名の長さ (バイト単位)。

IXFATYPN
副表の名前。

IXFATABL
IXFATABN フィールド内の表名の長さ (バイト単位)。

IXFATABN
表の名前。

IXFAPNDX
親副表の副表レコード索引。この副表が階層のルートである場合、このフィールドには値 -1 が入ります。

IXFASNDX
この副表を構成する列レコードの開始索引。

IXFAENDX
この副表を構成する列レコードの終了索引。

DB2 継続レコード

フィールド名	長さ	タイプ	備考
IXFARECL	006-BYTE	CHARACTER	レコード長
IXFARECT	001-BYTE	CHARACTER	レコード・タイプ = 'A'
IXFAPPID	012-BYTE	CHARACTER	application identifier = 'DB2 02.00'
IXFACTYP	001-BYTE	CHARACTER	アプリケーション固有のデータ・タイプ = 'C'
IXFADATE	008-BYTE	CHARACTER	'H' レコードから書き込まれた日付
IXFATIME	006-BYTE	CHARACTER	'H' レコードから書き込まれた時刻
IXFALAST	002-BYTE	SHORT INT	最後のディスケット・ボリューム番号
IXFATHIS	002-BYTE	SHORT INT	このディスケット・ボリューム番号
IXFANEXT	002-BYTE	SHORT INT	次のディスケット・ボリューム番号

ファイルが最終ボリュームでない限り、このレコードは、マルチボリュームの IXF ファイルの一部を成す各ファイルの末尾にあります。また、ファイルが先頭ボリュームでない限り、このレコードは、マルチボリュームの IXF ファイルの一部を成す各ファイルの先頭にもあります。このレコードの目的は、ファイル順序を記録しておくことです。 DB2 継続レコードには、以下のフィールドが入っています。

IXFARECL
レコード長標識。 PC/IXF レコードのうちこのレコード長標識より後の部分の長さをバイト単位で指定する整数値の 6 バイトの右寄せ文字表記 (合計レコード・サイズ - 6 バイト)。各 A レコードは、少なくとも IXFAPPID フィールドの全体を入れるのに十分な長さでなければなりません。

IXFARECT
IXF レコード・タイプ (このレコードの場合、A にセットされる)。これがアプリケーション・レコードであることを示します。アプリケーション ID によって暗黙指定されるデータの内容とフォーマットに関する特別の知識がないプログラムでは、このレコードは無視されます。

IXFAPPID
アプリケーション ID。これは、A レコードを作成したアプリケーションとして DB2 を識別します。

IXFACTYP
これがサブタイプ "C" の DB2 アプリケーション・レコードであることを指定します。

IXFADATE
ファイルの作成日付 (*yyyymmdd* の形式)。このフィールドの値は、IXFHDATE と同じでなければなりません。

IXFATIME
ファイルの作成時刻 (*hhmmss* の形式)。このフィールドの値は、IXFHTIME と同じでなければなりません。

IXFALAST
このフィールドは、リトル・エンディアン・フォーマットの 2 進数フィールドです。値は、IXFATHIS の値より 1 小さくなければなりません。

IXFATHIS
このフィールドは、リトル・エンディアン・フォーマットの 2 進数フィールドです。連続ボリューム上では、このフィールドの値も連続している必要があります。最初のボリュームでは 1 の値を持ちます。

IXFANEXT
このフィールドは、リトル・エンディアン・フォーマットの 2 進数フィールドです。レコードがファイルの先頭にない場合、値は、IXFATHIS 内の値よりも 1 大きくなければなりません。先頭にある場合、値はゼロでなければなりません。

DB2 終了レコード

フィールド名	長さ	タイプ	備考
IXFARECL	006-BYTE	CHARACTER	レコード長
IXFARECT	001-BYTE	CHARACTER	レコード・タイプ = 'A'
IXFAPPID	012-BYTE	CHARACTER	application identifier = 'DB2 02.00'
IXFAETYP	001-BYTE	CHARACTER	application specific data type = 'E'
IXFADATE	008-BYTE	CHARACTER	'H' レコードから書き込まれた日付
IXFATIME	006-BYTE	CHARACTER	'H' レコードから書き込まれた時刻

このレコードは、IXF ファイルの末尾にあるファイル終了マーカーです。 DB2 終了レコードには、以下のフィールドが入っています。

IXFARECL
レコード長標識。 PC/IXF レコードのうちこのレコード長標識より後の部分の長さをバイト単位で指定する整数値の 6 バイトの右寄せ文字表記 (合計レコード・サイズ - 6 バイト)。各 A レコードは、少なくとも IXFAPPID フィールドの全体を入れるのに十分な長さでなければなりません。

IXFARECT
IXF レコード・タイプ (このレコードの場合、A にセットされる)。これがアプリケーション・レコードであることを示します。アプリケーション ID によって暗黙指定されるデータの内容とフォーマットに関する特別の知識がないプログラムでは、このレコードは無視されます。

IXFAPPID
アプリケーション ID。これは、A レコードを作成したアプリケーションとして DB2 を識別します。

IXFAETYP
これがサブタイプ "E" の DB2 アプリケーション・レコードであることを指定します。

IXFADATE
ファイルの作成日付 (*yyyymmdd* の形式)。このフィールドの値は、IXFHDATE と同じでなければなりません。

IXFATIME
ファイルの作成時刻 (*hhmmss* の形式)。このフィールドの値は、IXFHTIME と同じでなければなりません。

DB2 ID レコード

フィールド名	長さ	タイプ	備考

```
----------     -------   ---------   -------------
IXFARECL       06-BYTE   CHARACTER   レコード長
IXFARECT       01-BYTE   CHARACTER   レコード・タイプ = 'A'
IXFAPPID       12-BYTE   CHARACTER   アプリケーション ID
IXFATYPE       01-BYTE   CHARACTER   アプリケーション固有のレコード・タイプ = 'S'
IXFADATE       08-BYTE   CHARACTER   アプリケーション・レコードの作成日
IXFATIME       06-BYTE   CHARACTER   アプリケーション・レコードの作成時刻
IXFACOLN       06-BYTE   CHARACTER   ID 列の列番号
IXFAITYP       01-BYTE   CHARACTER   常に生成される ('Y' または 'N')
IXFASTRT       33-BYTE   CHARACTER   ID 列の START AT 値
IXFAINCR       33-BYTE   CHARACTER   ID 列の INCREMENT BY 値
IXFACACH       10-BYTE   CHARACTER   ID 列の CACHE 値
IXFAMINV       33-BYTE   CHARACTER   ID 列の MINVALUE
IXFAMAXV       33-BYTE   CHARACTER   ID 列の MAXVALUE
IXFACYCL       01-BYTE   CHARACTER   ID 列の CYCLE ('Y' または 'N')
IXFAORDR       01-BYTE   CHARACTER   ID 列の ORDER ('Y' または 'N')
IXFARMRL       03-BYTE   CHARACTER   ID 列の注釈の長さ
IXFARMRK      254-BYTE   CHARACTER   ID 列の注釈の値
```

DB2 識別レコードには、以下のフィールドが入っています。

IXFARECL

レコード長標識。 PC/IXF レコードのうちこのレコード長標識より後の部分の長さをバイト単位で指定する整数値の 6 バイトの右寄せ文字表記 (合計レコード・サイズ - 6 バイト)。各 A レコードは、少なくとも IXFAPPID フィールドの全体を入れるのに十分な長さでなければなりません。

IXFARECT

IXF レコード・タイプ (このレコードの場合、A にセットされる)。これがアプリケーション・レコードであることを示します。アプリケーション ID によって暗黙指定されるデータの内容とフォーマットに関する特別の知識がないプログラムでは、このレコードは無視されます。

IXFAPPID

アプリケーション ID。これは、A レコードを作成したアプリケーションとして DB2 を識別します。

IXFATYPE

アプリケーション固有のレコード・タイプ。このフィールドの値は常に "S" でなければなりません。

IXFADATE

ファイルの作成日付 (*yyyymmdd* の形式)。このフィールドの値は、IXFHDATE と同じでなければなりません。

IXFATIME

ファイルの作成時刻 (*hhmmss* の形式)。このフィールドの値は、IXFHTIME と同じでなければなりません。

IXFACOLN

表内の ID 列の列番号。

IXFAITYP

ID 列のタイプ。 "Y" の値は、ID 列が常に GENERATED であることを示します。他のすべての値は、列が GENERATED BY DEFAULT タイプであることを意味すると解釈されます。

IXFASTRT

表の作成時に CREATE TABLE ステートメントで指定された ID 列の START AT 値。

IXFAINCR
表の作成時に CREATE TABLE ステートメントで指定された ID 列の INCREMENT BY 値。

IXFACACH
表の作成時に CREATE TABLE ステートメントで指定された ID 列の CACHE 値。 "1" の値は、NO CACHE オプションに対応します。

IXFAMINV
表の作成時に CREATE TABLE ステートメントで指定された ID 列の MINVALUE。

IXFAMAXV
表の作成時に CREATE TABLE ステートメントで指定された ID 列の MAXVALUE。

IXFACYCL
表の作成時に CREATE TABLE ステートメントで指定された ID 列の CYCLE 値。値 "Y" は CYCLE オプションに対応し、それ以外の値は NO CYCLE に対応します。

IXFAORDR
表の作成時に CREATE TABLE ステートメントで指定された ID 列の ORDER 値。値 "Y" は ORDER オプションに対応し、それ以外の値は NO ORDER に対応します。

IXFARMRL
IXFARMRK フィールド内の注釈の長さ (バイト単位)。

IXFARMRK
これは ID 列に関連付けられたユーザー入力の注釈です。これは単なる情報フィールドです。データベース・マネージャーは、データのインポート時にこのフィールドを使用しません。

DB2 SQLCA RECORD

フィールド名	長さ	タイプ	備考
IXFARECL	006-BYTE	CHARACTER	レコード長
IXFARECT	001-BYTE	CHARACTER	レコード・タイプ = 'A'
IXFAPPID	012-BYTE	CHARACTER	application identifier = 'DB2 02.00'
IXFAITYP	001-BYTE	CHARACTER	application specific data type = 'A'
IXFADATE	008-BYTE	CHARACTER	'H' レコードから書き込まれた日付
IXFATIME	006-BYTE	CHARACTER	'H' レコードから書き込まれた時刻
IXFASLCA	136-BYTE	variable	sqlca - SQL communications area

このタイプのレコードを 1 つ使用して、以後のインポート操作で IXF ファイルを使用して表を再作成できないことが示されます。詳しくは、IXFASLCA で戻されるメッセージと理由コードを参照してください。

DB2 SQLCA レコードには、以下のフィールドが入っています。

IXFARECL
レコード長標識。 PC/IXF レコードのうちこのレコード長標識より後の部分の長さをバイト単位で指定する整数値の 6 バイトの右寄せ文字表記 (合計レコード・サイズ - 6 バイト)。各「A」レコードは、少なくとも IXFAPPID フィールドの全体を入れるのに十分な長さでなければなりません。

IXFARECT

IXF レコード・タイプ (このレコードの場合、「A」にセットされる)。これがアプリケーション・レコードであることを示します。アプリケーション ID によって暗黙指定されるデータの内容とフォーマットに関する特別の知識がないプログラムでは、このレコードは無視されます。

IXFAPPID

アプリケーション ID。これは、「A」レコードを作成したアプリケーションとして DB2 を識別します。

IXFAITYP

これがサブタイプ「A」の DB2 アプリケーション・レコードであることを指定します。

IXFADATE

ファイルの作成日付 (*yyyymmdd* の形式)。このフィールドの値は、IXFHDATE と同じでなければなりません。

IXFATIME

ファイルの作成時刻 (*hhmmss* の形式)。このフィールドの値は、IXFHTIME と同じでなければなりません。

IXFASLCA

SQL 連絡域。SQL27984W 警告メッセージと、**IMPORT** コマンドで表を再作成するのに必要なすべての情報が IXF ファイル内に含まれていない理由を説明する理由コードが含まれます。

PC/IXF データ・タイプ:

表 23. PC/IXF データ・タイプ

名前	IXFCTYPE の値	説明
BIGINT	492	IXFTMFRM によって指定される形式の 8 バイトの整数。これは -9,223,372,036,854,775,808 から 9,223,372,036,854,775,807 の範囲の整数を表します。 IXFCSBCP および IXFCDBCP は無効であり 0 でなければなりません。 IXFCLENG は使用されず、ブランクを入れる必要があります。

表 23. PC/IXF データ・タイプ (続き)

名前	IXFCTYPE の値	説明
BLOB、CLOB	404、408	可変長文字ストリング。ストリングの最大長は、列記述子レコードの IXFCLENG フィールドに入れられ、それは 32,767 バイト以下です。ストリング自体の前には現行長標識が付きます。これは、ストリングの長さをバイト単位で指定する 4 バイト整数です。ストリングのコード・ページは、IXFCSBCP によって指定されるコード・ページです。 次の記述は BLOB にのみ適用されます。IXFCSBCP が 0 の場合、ストリングはビット・データであり、変換プログラムによって変換しないようにしてください。 次の記述は CLOB にのみ適用されます。IXFCDBCP がゼロ以外の場合、ストリングには IXFCDBCP によって指定されるコード・ページの 2 バイト文字も組み込むことができます。
BLOB_LOCATION_SPECIFIER および DBCLOB_LOCATION_SPECIFIER	960、964、968	固定長フィールド。255 バイトを超えてはなりません。 LOB ロケーション指定子 (LLS) は、 IXFCSBCP によって指定されるコード・ページにあります。 IXFCSBCP が 0 の場合、LLS はビット・データであり、変換プログラムによって変換しないようにしてください。 IXFCDBCP がゼロ以外の場合、ストリングには IXFCDBCP によって指定されるコード・ページの 2 バイト文字も組み込むことができます。 LLS の長さが IXFCLENG に保管されるため、元の LOB の実際の長さは失われます。LOB は LLS の長さで作成されるため、このタイプの列の入った PC/IXF ファイルを使用して LOB フィールドを再作成しないようにしてください。

表 23. PC/IXF データ・タイプ (続き)

名前	IXFCTYPE の値	説明
BLOB_FILE、 CLOB_FILE、 DBCLOB_FILE	916、920、924	*name_length* および *name* フィールドにデータが入れられた SQLFILE 構造体を収める固定長フィールド。ストリングの長さは、列記述子レコードの IXFCLENG フィールドに入れられ、それは 255 バイト以下です。ファイル名のコード・ページは、IXFCSBCP によって指定されるコード・ページです。IXFCDBCP がゼロ以外の場合、ファイル名には IXFCDBCP によって指定されるコード・ページの 2 バイト文字も組み込むことができます。 IXFCSBCP が 0 の場合、ファイル名はビット・データであり、変換プログラムによって変換しないようにしてください。 構造体の長さが IXFCLENG に保管されるため、元の LOB の実際の長さは失われます。LOB は *sql_lobfile_len* の長さで作成されるため、タイプ BLOB_FILE、 CLOB_FILE、または DBCLOB_FILE の列をもつ IXF ファイルを使用して LOB フィールドを再作成しないようにしてください。
CHAR	452	固定長文字ストリング。ストリングの長さは、列記述子レコードの IXFCLENG フィールドに入れられ、それは 254 バイト以下です。ストリングのコード・ページは、IXFCSBCP によって指定されるコード・ページです。 IXFCDBCP がゼロ以外の場合、ストリングには IXFCDBCP によって指定されるコード・ページの 2 バイト文字も組み込むことができます。 IXFCSBCP がゼロの場合、ストリングはビット・データであり、変換プログラムによって変換しないようにしてください。
DATE	384	グレゴリオ暦に準拠したポイント・イン・タイム。それぞれの日付は、国際標準化機構規格 (ISO) フォーマット (*yyyy-mm-dd*) の 10 バイトの文字ストリングです。年の部分の範囲は `0001` から `9999` です。月の部分の範囲は `01` から `12` です。日の部分の範囲は `01` から *n* です。ここで、*n* は月によって異なり、月の日数とうるう年に関する一般的な規則に従います。どの部分においても、先行ゼロは省略できません。 IXFCLENG は使用されず、ブランクを入れる必要があります。DATE 内の有効な文字は、すべての PC ASCII コード・ページで不変です。そのため、IXFCSBCP および IXFCDBCP は有効ではなく、0 でなければなりません。

表 23. PC/IXF データ・タイプ (続き)

名前	IXFCTYPE の値	説明
DBCLOB	412	2 バイト文字の可変長ストリング。列記述子レコードの IXFCLENG フィールドは、ストリング内の 2 バイト文字の最大文字数を指定するもので、16,383 以下です。ストリング自体の前に現行長標識が付きます。これは、ストリングの長さを 2 バイト文字の文字数で指定する 4 バイト整数です (つまりこの整数の値は、バイト単位のストリング長の半分の値です)。ストリングのコード・ページは、C レコードの IXFCDBCP によって指定される DBCS コード・ページです。ストリングの内容は 2 バイト文字データだけなので、IXFCSBCP は 0 でなければなりません。ストリングを囲むシフトインまたはシフトアウト文字はありません。
DECIMAL	484	精度が P (列記述子レコードの IXFCLENG の最初の 3 バイトによって指定される) で、位取りが S (IXFCLENG の最後の 2 バイトによって指定される) であるパック 10 進数。パック 10 進数の長さは (P+2)/2 です (バイト単位)。精度は 1 から 31 の範囲の奇数でなければなりません。パック 10 進数は、IXFTMFRM によって指定される内部フォーマットになっています。 IXFTMFRM では、PC のパック 10 進数が System/370 のパック 10 進数と同じになるように定義されます。 IXFCSBCP および IXFCDBCP は無効であり 0 でなければなりません。
DECFLOAT	996	10 進浮動小数点値は、小数点付きの IEEE 754r 数です。小数点の位置は、それぞれの 10 進浮動小数点値に格納されます。 10 進浮動小数点数の範囲は、精度が 16 桁または 34 桁で、指数範囲はそれぞれ 10-383 から 10+384 までと 10-6143 から 10+6144 までです。16 桁値の格納される長さは 8 バイトで、34 桁値の格納される長さは 16 バイトです。
FLOATING POINT	480	長精度 (8 バイト) または短精度 (4 バイト) 浮動小数点数。これは、IXFCLENG が 8 または 4 のどちらにセットされているかによって決まります。データは、IXFTMFRM によって指定される内部マシン・フォーマットです。 IXFCSBCP および IXFCDBCP は無効であり 0 でなければなりません。 4 バイトの浮動小数点数は、データベース・マネージャーではサポートされません。

表 23. PC/IXF データ・タイプ (続き)

名前	IXFCTYPE の値	説明
GRAPHIC	468	2 バイト文字の固定長ストリング。列記述子レコードの IXFCLENG フィールドは、ストリング内の 2 バイト文字の文字数を指定するもので、127 以下です。ストリングの実際の長さ (バイト単位) は、IXFCLENG フィールドの値の 2 倍です。ストリングのコード・ページは、C レコードの IXFCDBCP によって指定される DBCS コード・ページです。ストリングの内容は 2 バイト文字データだけなので、IXFCSBCP は 0 でなければなりません。ストリングを囲むシフトインまたはシフトアウト文字はありません。
INTEGER	496	IXFTMFRM によって指定される形式の 4 バイト整数。これは -2,147,483,648 から +2,147,483,647 の範囲の整数を表します。IXFCSBCP および IXFCDBCP は無効であり 0 でなければなりません。 IXFCLENG は使用されず、ブランクを入れる必要があります。
LONGVARCHAR	456	可変長文字ストリング。ストリングの最大長は、列記述子レコードの IXFCLENG フィールドに入れられ、それは 32,767 バイト以下です。ストリング自体の前には現行長標識が付きます。これは、ストリングの長さをバイト単位で指定する 2 バイトの整数です。ストリングのコード・ページは、IXFCSBCP によって指定されるコード・ページです。IXFCDBCP がゼロ以外の場合、ストリングには IXFCDBCP によって指定されるコード・ページの 2 バイト文字も組み込むことができます。 IXFCSBCP がゼロの場合、ストリングはビット・データであり、変換プログラムによって変換しないようにしてください。

表 23. PC/IXF データ・タイプ (続き)

名前	IXFCTYPE の値	説明
LONG VARGRAPHIC	472	2 バイト文字の可変長ストリング。列記述子レコードの IXFCLENG フィールドは、ストリングの 2 バイト文字の最大文字数を指定するもので、16,383 以下です。ストリング自体の前に現行長標識が付きます。これは、ストリングの長さを 2 バイト文字の文字数で指定する 2 バイトの整数です (つまりこの整数の値は、バイト単位のストリング長の半分の値です)。ストリングのコード・ページは、C レコードの IXFCDBCP によって指定される DBCS コード・ページです。ストリングの内容は 2 バイト文字データだけなので、IXFCSBCP は 0 でなければなりません。ストリングを囲むシフトインまたはシフトアウト文字はありません。
SMALLINT	500	IXFTMFRM によって指定される形式の 2 バイトの整数。これは -32,768 から +32,767 の範囲の整数を表します。 IXFCSBCP および IXFCDBCP は無効であり 0 でなければなりません。 IXFCLENG は使用されず、ブランクを入れる必要があります。
TIME	388	24 時間制によるポイント・イン・タイム。それぞれの時刻は、ISO フォーマット (*hh.mm.ss*) の 8 バイトのストリングです。時の部分の範囲は `00` から `24` で、その他の部分の範囲は `00` から `59` です。時が `24` の場合、その他の部分は `00` です。最小の時刻は `00.00.00`、最大の時刻は `24.00.00` です。どの部分においても、先行ゼロは省略できません。 IXFCLENG は使用されず、ブランクを入れる必要があります。 TIME 内の有効な文字は、すべての PC ASCII コード・ページで不変です。そのため、IXFCSBCP および IXFCDBCP は有効ではなく、0 でなければなりません。

表 23. PC/IXF データ・タイプ (続き)

名前	IXFCTYPE の値	説明
TIMESTAMP	392	小数位の秒数の精度を持つ日時。各タイム・スタンプは、*yyyy-mm-dd-hh.mm.ss.nnnnnn* (年、月、日、時、分、秒、秒の小数部分) の形式の文字ストリングです。バージョン 9.7から、列記述子レコードの IXFCLENG フィールドにはタイム・スタンプの精度が入り、値は 12 以下であるようになりました。バージョン 9.7より前では、IXFCLENG は使用されず、ブランクを入れる必要があります。 TIMESTAMP 内の有効な文字は、すべての PC ASCII コード・ページで不変です。そのため、IXFCSBCP および IXFCDBCP は有効ではなく、0 でなければなりません。
VARCHAR	448	可変長文字ストリング。ストリングの最大長 (バイト単位) は、列記述子レコードの IXFCLENG フィールドに入れられ、それは 254 バイト以下です。ストリング自体の前には現行長標識が付きます。これは、ストリングの長さをバイト単位で指定する 2 バイトの整数です。ストリングのコード・ページは、IXFCSBCP によって指定されるコード・ページです。 IXFCDBCP がゼロ以外の場合、ストリングには IXFCDBCP によって指定されるコード・ページの 2 バイト文字も組み込むことができます。 IXFCSBCP がゼロの場合、ストリングはビット・データであり、変換プログラムによって変換しないようにしてください。
VARGRAPHIC	464	2 バイト文字の可変長ストリング。列記述子レコードの IXFCLENG フィールドは、ストリング内の 2 バイト文字の最大文字数を指定するもので、127 以下です。ストリング自体の前に現行長標識が付きます。これは、ストリングの長さを 2 バイト文字の文字数で指定する 2 バイトの整数です (つまりこの整数の値は、バイト単位のストリング長の半分の値です)。ストリングのコード・ページは、C レコードの IXFCDBCP によって指定される DBCS コード・ページです。ストリングの内容は 2 バイト文字データだけなので、IXFCSBCP は 0 でなければなりません。ストリングを囲むシフトインまたはシフトアウト文字はありません。

PC/IXF 文字または GRAPHIC 列では、IXFCSBCP 値と IXFCDBCP 値の一部の組み合わせは無効です。 IXFCSBCP と IXFCDBCP の組み合わせが無効である PC/IXF 文字または GRAPHIC 列は、無効なデータ・タイプです。

表 24. 有効な PC/IXF データ・タイプ

PC/IXF データ・タイプ	有効な (IXFCSBCP,IXFCDBCP) 対	無効な (IXFCSBCP,IXFCDBCP) 対
CHAR、VARCHAR、または LONG VARCHAR	(0,0)、(x,0)、または (x,y)[1]	(0,y)[1]
BLOB	(0,0)	(x,0)、(0,y)、または (x,y)[1]
CLOB	(x,0)、(x,y)[1]	(0,0)、(0,y)[1]
GRAPHIC、VARGRAPHIC、LONG VARGRAPHIC、または DBCLOB	(0,y)[1]	(0,0)、(x,0)、または (x,y)[1]

注: [1] x も y も 0 ではありません。

PC/IXF のデータ・タイプの説明:

表 25. PC/IXF ファイル・フォーマットで受け入れられるデータ・タイプの形式

データ・タイプ	エクスポート・ユーティリティーによって作成されるファイルの形式	インポート・ユーティリティーにとって受け入れ可能な形式
BIGINT	データベース列と同じ BIGINT 列が作成されます。	すべての数値タイプ (SMALLINT、INTEGER、 BIGINT、 DECIMAL、または FLOAT) の列が受け入れられます。 -9,223,372,036,854,775,808 から 9,223,372,036,854,775,807 の範囲外の値があれば、その特定の値に関してはリジェクトされます。
BLOB	PC/IXF BLOB 列が作成されます。データベース列の最大長、SBCS CPGID 値、および DBCS CPGID 値が列記述子レコードにコピーされます。	次の場合、 PC/IXF CHAR、 VARCHAR、 LONG VARCHAR、 BLOB、 BLOB_FILE、または BLOB_LOCATION_SPECIFIER 列が受け入れ可能です。 • データベース列が FOR BIT DATA としてマークされている場合。 • PC/IXF 列の 1 バイト・コード・ページ値がデータベース列の SBCS CPGID と等しく、PC/IXF 列の 2 バイト・コード・ページ値が 0 またはデータベース列の DBCS CPGID と等しい場合。 PC/IXF GRAPHIC、VARGRAPHIC、または LONG VARGRAPHIC の BLOB 列も受け入れ可能です。 PC/IXF 列が固定長である場合、その長さはデータベース列の最大長と互換性がなければなりません。

表 25. *PC/IXF* ファイル・フォーマットで受け入れられるデータ・タイプの形式 (続き)

データ・タイプ	エクスポート・ユーティリティーによって作成されるファイルの形式	インポート・ユーティリティーにとって受け入れ可能な形式
CHAR	PC/IXF CHAR 列が作成されます。データベース列の長さ、SBCS CPGID 値、および DBCS CPGID 値が PC/IXF 列記述子レコードにコピーされます。	次の場合、PC/IXF CHAR、 VARCHAR、または LONG VARCHAR 列が受け入れ可能です。 • データベース列が FOR BIT DATA としてマークされている場合。 • PC/IXF 列の 1 バイト・コード・ページ値がデータベース列の SBCS CPGID と等しく、PC/IXF 列の 2 バイト・コード・ページ値が 0 またはデータベース列の DBCS CPGID と等しい場合。 データベース列が FOR BIT DATA としてマークされている場合、PC/IXF GRAPHIC、VARGRAPHIC、または LONG VARGRAPHIC 列も受け入れ可能です。どちらの場合でも、PC/IXF 列が固定長なら、その長さはデータベース列の長さと互換性がなければなりません。データは、必要な場合右側に 1 バイト・スペース (x'20') が埋められます。
CLOB	PC/IXF CLOB 列が作成されます。データベース列の最大長、SBCS CPGID 値、および DBCS CPGID 値が列記述子レコードにコピーされます。	PC/IXF 列の単一バイト・コード・ページの値がデータベース列の SBCS CPGID と等しく、 PC/IXF 列の 2 バイト・コード・ページの値がゼロか、データベース列の DBCS CPGID と等しい場合には、PC/IXF CHAR、VARCHAR、 LONG VARCHAR、 CLOB、CLOB_FILE、または CLOB_LOCATION_SPECIFIER 列が受け入れ可能です。 PC/IXF 列が固定長である場合、その長さはデータベース列の最大長と互換性がなければなりません。
DATE	データベース列と同じ DATE 列が作成されます。	タイプ DATE の PC/IXF 列は通常の入力です。インポート・ユーティリティーは、長さに互換性がないものを除くすべての文字タイプの列を受け入れようとします。 PC/IXF ファイル内の文字タイプの列の内容は、ターゲット・データベースの地域コードと互換性のあるフォーマットの日付でなければなりません。

表 25. PC/IXF ファイル・フォーマットで受け入れられるデータ・タイプの形式 (続き)

データ・タイプ	エクスポート・ユーティリティーによって作成されるファイルの形式	インポート・ユーティリティーにとって受け入れ可能な形式
DBCLOB	PC/IXF DBCLOB 列が作成されます。データベース列の最大長、SBCS CPGID 値、および DBCS CPGID 値が列記述子レコードにコピーされます。	PC/IXF 列の 2 バイト・コード・ページ値がデータベース列と等しい場合は、 PC/IXF GRAPHIC、 VARGRAPHIC、 LONG VARGRAPHIC、 DBCLOB、 DBCLOB_FILE、または DBCLOB_LOCATION_SPECIFIER 列が受け入れ可能です。 PC/IXF 列が固定長である場合、その長さはデータベース列の最大長と互換性がなければなりません。
DECIMAL	データベース列と同じ DECIMAL 列が作成されます。列の精度と位取りが列記述子レコードに保管されます。	すべての数値タイプ (SMALLINT、 INTEGER、 BIGINT、 DECIMAL、または FLOAT) の列が受け入れられます。インポート先の DECIMAL 列の範囲外の値があれば、その特定の値に関してはリジェクトされます。
DECFLOAT	データベース列と同じ DECFLOAT 列が作成されます。列の精度が列記述子レコードに保管されます。	SMALLINT、INTEGER、BIGINT (DECFLOAT(34) のみ)、 DECIMAL、FLOAT、REAL、DOUBLE、または DECFLOAT(16) (DECFLOAT(34) のみ) 型の列が受け入れられます。DECFLOAT の場合は他の数値列タイプも有効ですが、ターゲットの精度に適合しない場合は、丸められます。
FLOAT	データベース列と同じ FLOAT 列が作成されます。	すべての数値タイプ (SMALLINT、 INTEGER、 BIGINT、 DECIMAL、または FLOAT) の列が受け入れられます。すべての値は範囲内です。
GRAPHIC (DBCS のみ)	PC/IXF GRAPHIC 列が作成されます。データベース列の長さ、SBCS CPGID 値、および DBCS CPGID 値が列記述子レコードにコピーされます。	PC/IXF 列の 2 バイト・コード・ページ値がデータベース列と同じである場合は、 PC/IXF GRAPHIC、 VARGRAPHIC、または LONG VARGRAPHIC 列が受け入れ可能です。PC/IXF 列が固定長である場合、その長さはデータベース列の長さと互換性がなければなりません。データは、必要な場合右側に 2 バイト・スペース (x'8140') が埋められます。
INTEGER	データベース列と同じ INTEGER 列が作成されます。	すべての数値タイプ (SMALLINT、 INTEGER、 BIGINT、 DECIMAL、または FLOAT) の列が受け入れられます。 -2,147,483,648 から 2,147,483,647 の範囲外の整数があれば、それに関してはリジェクトされます。

表 25. *PC/IXF* ファイル・フォーマットで受け入れられるデータ・タイプの形式 (続き)

データ・タイプ	エクスポート・ユーティリティーによって作成されるファイルの形式	インポート・ユーティリティーにとって受け入れ可能な形式
LONG VARCHAR	PC/IXF LONG VARCHAR 列が作成されます。データベース列の最大長、 SBCS CPGID 値、および DBCS CPGID 値が列記述子レコードにコピーされます。	次の場合、PC/IXF CHAR、 VARCHAR、または LONG VARCHAR 列が受け入れ可能です。 • データベース列が FOR BIT DATA としてマークされている場合。 • PC/IXF 列の 1 バイト・コード・ページ値がデータベース列の SBCS CPGID と等しく、PC/IXF 列の 2 バイト・コード・ページ値が 0 またはデータベース列の DBCS CPGID と等しい場合。 データベース列が FOR BIT DATA としてマークされている場合、PC/IXF GRAPHIC、VARGRAPHIC、または LONG VARGRAPHIC 列も受け入れ可能です。どちらの場合でも、PC/IXF 列が固定長なら、その長さはデータベース列の最大長と互換性がなければなりません。
LONG VARGRAPHIC (DBCS のみ)	PC/IXF LONG VARGRAPHIC 列が作成されます。データベース列の最大長、 SBCS CPGID 値、および DBCS CPGID 値が列記述子レコードにコピーされます。	PC/IXF 列の 2 バイト・コード・ページ値がデータベース列と同じである場合は、 PC/IXF GRAPHIC、 VARGRAPHIC、または LONG VARGRAPHIC 列が受け入れ可能です。PC/IXF 列が固定長である場合、その長さはデータベース列の最大長と互換性がなければなりません。
SMALLINT	データベース列と同じ SMALLINT 列が作成されます。	すべての数値タイプ (SMALLINT、 INTEGER、 BIGINT、 DECIMAL、または FLOAT) の列が受け入れられます。 -32,768 から 32,767 の範囲外の値があれば、その特定の値に関してはリジェクトされます。
TIME	データベース列と同じ TIME 列が作成されます。	タイプ TIME の PC/IXF 列は通常の入力です。インポート・ユーティリティーは、長さに互換性がないものを除くすべての文字タイプの列を受け入れようとします。 PC/IXF ファイル内の文字タイプの列の内容は、ターゲット・データベースの地域コードと互換性のあるフォーマットの時刻データでなければなりません。

表 25. PC/IXF ファイル・フォーマットで受け入れられるデータ・タイプの形式 (続き)

データ・タイプ	エクスポート・ユーティリティーによって作成されるファイルの形式	インポート・ユーティリティーにとって受け入れ可能な形式
TIMESTAMP	データベース列と同じ TIMESTAMP 列が作成されます。	タイプ TIMESTAMP の PC/IXF 列は通常の入力です。インポート・ユーティリティーは、長さに互換性がないものを除くすべての文字タイプの列を受け入れようとします。PC/IXF ファイルの文字タイプの列に入っているデータは、タイム・スタンプの入力フォーマットになっていなければなりません。
VARCHAR	データベース列の最大長が = 254 の場合は、 PC/IXF VARCHAR 列が作成されます。データベース列の最大長が > 254 の場合は、 PC/IXF LONG VARCHAR 列が作成されます。データベース列の最大長、 SBCS CPGID 値、および DBCS CPGID 値が列記述子レコードにコピーされます。	次の場合、PC/IXF CHAR、 VARCHAR、または LONG VARCHAR 列が受け入れ可能です。 • データベース列が FOR BIT DATA としてマークされている場合。 • PC/IXF 列の 1 バイト・コード・ページ値がデータベース列の SBCS CPGID と等しく、PC/IXF 列の 2 バイト・コード・ページ値が 0 またはデータベース列の DBCS CPGID と等しい場合。 データベース列が FOR BIT DATA としてマークされている場合、PC/IXF GRAPHIC、VARGRAPHIC、または LONG VARGRAPHIC 列も受け入れ可能です。どちらの場合でも、PC/IXF 列が固定長なら、その長さはデータベース列の最大長と互換性がなければなりません。
VARGRAPHIC (DBCS のみ)	データベース列の最大長が = 127 の場合は、 PC/IXF VARGRAPHIC 列が作成されます。データベース列の最大長が > 127 の場合は、 PC/IXF LONG VARGRAPHIC 列が作成されます。データベース列の最大長、 SBCS CPGID 値、および DBCS CPGID 値が列記述子レコードにコピーされます。	PC/IXF 列の 2 バイト・コード・ページ値がデータベース列と同じである場合は、 PC/IXF GRAPHIC、 VARGRAPHIC、または LONG VARGRAPHIC 列が受け入れ可能です。PC/IXF 列が固定長である場合、その長さはデータベース列の最大長と互換性がなければなりません。

PC/IXF ファイルのデータベースへのインポートを制御する一般規則:

データベース・マネージャーのインポート・ユーティリティーでは、SBCS または DBCS 環境での PC/IXF ファイルのインポート時に、以下の規則が適用されます。

- インポート・ユーティリティーは、PC/IXF フォーマットのファイル (IXFHID = 'IXF') のみ受け入れます。その他のフォーマットの IXF ファイルはインポートできません。
- インポート・ユーティリティーは、1024 個より多くの列が入っている PC/IXF ファイルをリジェクトします。
- IXF 形式にエクスポートするときに、ID が、IXF 形式でサポートされている最大サイズを超えていると、エクスポート操作は成功しますが、その結果として生成されるデータ・ファイルを後から CREATE モードのインポート操作で使用することはできません。SQL27984W が戻されます。

  注: IMPORT コマンドの CREATE オプションと REPLACE_CREATE オプションは非推奨で、将来のリリースでは廃止される可能性があります。
- PC/IXF H レコードの IXFHSBCP の値は、SBCS CPGID と等しくなければなりません。あるいは、IXFHSBCP/IXFHDBCP と、ターゲット・データベースの SBCS/DBCS CPGID との間の変換表がなければなりません。 IXFHDBCP の値は '00000' か、またはターゲット・データベースの DBCS CPGID と等しくなければなりません。これらの条件のいずれも満たされない場合、インポート・ユーティリティーは、 FORCEIN オプションが指定されない限り、PC/IXF ファイルをリジェクトします。
- 無効なデータ・タイプ - 新しい表

  PC/IXF ファイルの*新しい* 表へのインポートは、IMPORT コマンドの CREATE または REPLACE_CREATE キーワードによって指定されます。新しい表へのインポートにおいて無効なデータ・タイプの PC/IXF 列が選択されると、インポート・ユーティリティーは終了します。 PC/IXF ファイル全体がリジェクトされ、表は作成されず、データはインポートされません。
- 無効なデータ・タイプ - 既存の表

  PC/IXF ファイルの*既存の* 表へのインポートは、**IMPORT** コマンドの INSERT、INSERT_UPDATE、REPLACE、または REPLACE_CREATE キーワードによって指定されます。既存の表へのインポートにおいて無効なデータ・タイプの PC/IXF 列が選択されると、次のいずれかの処理が実行されます。
  - ターゲット表の列が NULL 可能である場合は、無効な PC/IXF 列のすべての値が無視され、表の列の値は NULL にセットされます。
  - ターゲット表の列が NULL 可能でない場合、インポート・ユーティリティーは終了します。 PC/IXF ファイル全体がリジェクトされ、データはインポートされません。既存の表は未変更のままです。
- 新しい表へのインポートにおいて、NULL 可能な PC/IXF 列は NULL 可能なデータベース列を生成し、NULL 可能でない PC/IXF 列は NULL 可能でないデータベース列を生成します。
- NULL 可能でない PC/IXF 列を、NULL 可能なデータベース列にインポートすることができます。
- NULL 可能 PC/IXF 列を、NULL 可能でないデータベース列にインポートすることができます。 PC/IXF 列内で NULL 値が検出されると、インポート・ユーティリティーは NULL 値を備えた PC/IXF 行のすべての列の値をリジェクトし (行全体がリジェクトされ)、次の PC/IXF 行から処理が継続されます。つまり、

NULL のターゲット表の列が NULL 可能でない場合、その NULL 値の入った PC/IXF 行からはデータがインポートされません。

- 互換性のない列 - 新しい表

  新しい データベース表へのインポート中に、ターゲット・データベース列と互換性がない PC/IXF 列が選択されると、インポート・ユーティリティーは終了します。 PC/IXF ファイル全体がリジェクトされ、表は作成されず、データはインポートされません。

  注: IMPORT の FORCEIN オプションを使用すると、互換性のある列の範囲が広がります。

- 互換性のない列 - 既存の表

  既存の データベース表へのインポート中に、ターゲット・データベース列と互換性がない PC/IXF 列が選択されると、次の 2 つの処理のうちいずれかが実行されます。

  – ターゲット表の列が NULL 可能である場合は、PC/IXF 列のすべての値が無視され、表の列の値は NULL にセットされます。
  – ターゲット表の列が NULL 可能でない場合、インポート・ユーティリティーは終了します。 PC/IXF ファイル全体がリジェクトされ、データはインポートされません。既存の表は未変更のままです。

  注: IMPORT の FORCEIN オプションを使用すると、互換性のある列の範囲が広がります。

- 無効な値

  インポート中に、ターゲット・データベース列にとって無効な PC/IXF 列値が検出されると、インポート・ユーティリティーは、無効な値の入った PC/IXF 行のすべての列の値をリジェクトし (行全体がリジェクトされ)、次の PC/IXF 行から処理が継続されます。

PC/IXF ファイルのデータベースへのインポートを制御するデータ・タイプ固有の規則:

- 有効な PC/IXF 数値列は、互換性のある任意のデータベース数値列にインポートできます。 4 バイトの浮動小数点データが入っている PC/IXF 列は無効なデータ・タイプであるため、インポートできません。
- データベースの日付/時刻列は、対応する PC/IXF 日付/時刻列 (DATE、TIME、および TIMESTAMP) からの値、および列の長さおよび値の互換性制限に従っている PC/IXF 文字タイプ列 (CHAR、VARCHAR、および LONG VARCHAR) からの値を受け入れることができます。
- 有効な PC/IXF 文字タイプ列 (CHAR、VARCHAR、または LONG VARCHAR) は、FOR BIT DATA としてマークされている*既存の* データベース文字タイプ列に常にインポートできます。それ以外の場合、
  – IXFCSBCP と SBCS CPGID は一致していなければなりません。
  – IXFCSBCP/IXFCDBCP と SBCS/DBCS のための変換表がなければなりません。
  – 1 つのセットがすべて 0 (FOR BIT DATA) でなければなりません。

IXFCSBCP が 0 でない場合、IXFCDBCP の値は 0、またはターゲット・データベース列の DBCS CPGID の値と等しくなければなりません。

これらの条件のいずれかが満たされていない場合、PC/IXF とデータベース列には互換性がありません。

有効な PC/IXF 文字タイプ列を*新しい* データベース表にインポートする場合、IXFCSBCP の値は 0 またはデータベースの SBCS CPGID と等しくなければなりません。あるいは変換表がなければなりません。 IXFCSBCP が 0 の場合は、IXFCDBCP も 0 でなければなりません (そうでなければ、PC/IXF 列は無効なデータ・タイプです)。この場合、IMPORT は、新しい表の中に FOR BIT DATA としてマークされた文字タイプの列を作成します。 IXFCSBCP が 0 でなくデータベースの SBCS CPGID と等しい場合、 IXFCDBCP の値は 0 またはデータベースの DBCS CPGID でなければなりません。この場合、ユーティリティーは、新しい表の中に SBCS および DBCS CPGID 値がデータベースと等しい文字タイプの列を作成します。これらの条件が満たされていない場合、PC/IXF とデータベース列には互換性がありません。

FORCEIN オプションを使用すると、コード・ページの同等性チェックをオーバーライドすることができます。しかし、IXFCSBCP が 0 で IXFCDBCP が 0 でない PC/IXF 文字タイプ列は無効なデータ・タイプであり、FORCEIN が指定されていてもインポートできません。

- 有効な PC/IXF 文字タイプ列 (GRAPHIC、VARGRAPHIC、または LONG VARGRAPHIC) は、 FOR BIT DATA としてマークされている*既存の* データベース文字タイプ列に常にインポートできます。 FORCEIN オプションを使用すると、この制限を緩和することができます。しかし、IXFCSBCP が 0 でないか、または IXFCDBCP が 0 の PC/IXF GRAPHIC 列は無効なデータ・タイプであり、FORCEIN が指定されていてもインポートできません。

  有効な PC/IXF GRAPHIC 列をデータベースの GRAPHIC 列にインポートする場合、 IXFCDBCP の値はターゲット・データベース列の DBCS CPGID と等しくなければなりません (つまり 2 つの列の 2 バイト・コード・ページが一致していなければなりません)。
- PC/IXF ファイルの既存のデータベース表へのインポート中に固定長ストリング列 (CHAR または GRAPHIC) が選択され、その長さがターゲット列の最大長より長い場合、それらの列には互換性がありません。
- PC/IXF ファイルの既存のデータベース表へのインポート中に、可変長ストリング列 (VARCHAR、LONG VARCHAR、VARGRAPHIC、または LONG VARGRAPHIC) が選択され、その長さがターゲット列の最大長より長い場合、それらの列には互換性があります。個々の値は、データベース・マネージャー INSERT 互換規則に従って、ターゲット・データベース列にとって長すぎる PC/IXF 値は無効です。
- 固定長のデータベースの文字 タイプの列 (つまり CHAR 列) にインポートされる PC/IXF 値は、必要な場合値の長さがデータベース列と等しくなるよう右側に 1 バイト・スペース (0x20) が埋められます。固定長のデータベースの *GRAPHIC* タイプの列 (つまり GRAPHIC 列) にインポートされる PC/IXF 値は、必要な場合値の長さがデータベース列と等しくなるよう右側に 2 バイト・スペース (0x8140) が埋められます。

- PC/IXF VARCHAR 列の最大長は 254 バイトであるため、最大長が *n* (254 *n* 4001) の PC/IXF VARCHAR 列は、最大長が *n* のデータベース LONG VARCHAR 列にエクスポートしなければなりません。
- PC/IXF LONG VARCHAR 列の最大長は 32,767 バイトで、データベース LONG VARCHAR 列には 32,700 バイトの最大長制限がありますが、長さが 32,700 バイトを超える (ただし 32,768 バイトよりも小さい) PC/IXF LONG VARCHAR 列は有効であり、データベース LONG VARCHAR 列にインポートできます。ただし、データが失われる可能性があります。
- PC/IXF VARGRAPHIC 列の最大長は 127 バイトであるため、最大長が *n* (127 *n* 2001) の PC/IXF VARGRAPHIC 列は、最大長が *n* のデータベース LONG VARGRAPHIC 列にエクスポートしなければなりません。
- PC/IXF LONG VARGRAPHIC 列の最大長は 16,383 バイトで、データベース LONG VARGRAPHIC 列には 16,350 バイトの最大長制限がありますが、長さが 16,350 バイトを超える (ただし 16,384 バイトよりも小さい) PC/IXF LONG VARGRAPHIC 列は有効であり、データベース LONG VARGRAPHIC 列にインポートできます。ただし、データが失われる可能性があります。

FORCEIN オプションを使用せずに PC/IXF ファイルを新しいデータベース表または既存のデータベース表にインポートする場合について、表 26 および 280 ページの表 27 にまとめます。

表 26. FORCEIN オプションを使用しない場合の PC/IXF ファイルのインポートのサマリー - 数値タイプ

PC/IXF の列データ・タイプ	**データベースの列データ・タイプ**					
	SMALL INT	**INT**	**BIGINT**	**DEC**	**DFP**	**FLT**
-SMALLINT	N					
	E	E	E	E[a]	E	E
-INTEGER		N				
	E[a]	E	E	E[a]	E	E
-BIGINT			N			
	E[a]	E[a]	E	E[a]	E	E
-DECIMAL				N		
	E[a]	E[a]	E[a]	E[a]	E	E
-DECFLOAT					N	
	E[a]	E[a]	E[a]	E[a]	E	E[a]
-FLOAT						N
	E[a]	E[a]	E[a]	E[a]	E	E

[a] ターゲットの数値データ・タイプの範囲内にない値は、その特定の値に関してリジェクトされます。

表 27. FORCEIN オプションを使用しない場合の PC/IXF ファイルのインポートのサマリー - 文字、グラフィック、および日時の各タイプ

PC/IXF の列データ・タイプ	データベースの列データ・タイプ						
	(0,0)	(SBCS, 0)[d]	(SBCS, DBCS)[b]	GRAPH[b]	DATE	TIME	TIME STAMP
-(0,0)	N						
	E				E[c]	E[c]	E[c]
-(SBCS,0)		N	N				
	E	E	E		E[c]	E[c]	E[c]
-(SBCS, DBCS)			N		E[c]	E[c]	E[c]
	E		E				
-GRAPHIC				N			
	E			E			
-DATE					N		
					E		
-TIME						N	
						E	
-TIME STAMP							N
							E

[b] データ・タイプは DBCS 環境でのみ使用可能です。
[c] 有効な日付または時刻の値でない値は、その特定の値に関してリジェクトされます。
[d] データ・タイプは DBCS 環境では使用不可です。

注:

1. この表は、すべての有効な PC/IXF およびデータベース・マネージャー・データ・タイプの一覧表です。 PC/IXF 列をデータベース列にインポートできる場合は、 PC/IXF データ・タイプの行とデータベース・マネージャー・データ・タイプの列が交差する位置のセルに文字が示されています。 'N' は、ユーティリティーが新しいデータベース表を作成することを示します (示されているデータ・タイプのデータベース列が作成されます)。 'E' は、ユーティリティーが既存のデータベース表にデータをインポートすることを示します (示されているデータ・タイプのデータベース列は有効なターゲットです)。
2. 文字ストリング・データ・タイプは、コード・ページ属性によって区別されます。これらの属性は、(SBCS,DBCS) の順序対として示されています。ここで、
   - SBCS は 0、または文字データ・タイプの 1 バイト・コード・ページ属性の非 0 値を示します。
   - DBCS は 0、または文字データ・タイプの 2 バイト・コード・ページ属性の非 0 値を示します。
3. この表で文字タイプの PC/IXF 列が文字タイプのデータベース列にインポートできることが示されている場合、それらのコード・ページ属性の対の値はコード・ページの同等性を制御する規則を満たします。

PC/IXF およびバージョン 0 の System/370 IXF の相違:

以下に、PC/IXF (データベース・マネージャーによって使用される) と、バージョン 0 System/370 IXF (いくつかのホスト・データベース製品によって使用される) の間の相違点について説明します。

- PC/IXF ファイルは、EBCDIC 指向ではなく ASCII です。 PC/IXF ファイルでは、H レコードの新しいコード・ページ ID や列記述子レコードでの実際のコード・ページ値の使用を含め、コード・ページ識別機能が大幅に拡張されています。さらに、文字データの列を FOR BIT DATA としてマークするためのメカニズムもあります。 PC/IXF ファイル・フォーマットと、その他の IXF またはデータベース・ファイル・フォーマットとの間の変換では、FOR BIT DATA 列内の値のコード・ページ変換ができないため、 FOR BIT DATA 列には特別な意義があります。
- マシン・データ・フォーマットのみ可能です。つまり、IXFTFORM フィールドの内容は常に M でなければなりません。さらに、マシン・データは PC 形式でなければなりません。つまり、IXFTMFRM フィールドの値は PC でなければなりません。したがって、PC/IXF データ・レコードのデータ部分の整数、浮動小数点数、および 10 進数は PC 形式でなければなりません。
- PC/IXF ファイルにおいて、H レコードの後であればどこにでもアプリケーション (A) レコードを入れることができます。それらは、IXFHHCNT フィールドの値の計算ではカウントされません。
- すべての PC/IXF レコードはレコード長標識で始まります。これは、PC/IXF レコードのうちレコード長標識自体を除く長さの整数値を 6 バイトの文字で表記したものです (つまり合計レコード長 - 6 バイト)。レコード長フィールドの目的は、PC プログラムがレコード境界を識別できるようにすることです。
- 可変長データによるストレージの節約を活用するため、またフィールドが複数のレコードに分割されている場合の複雑な処理を回避するため、 PC/IXF では、バージョン 0 の IXF X レコードはサポートされませんが、D レコード ID はサポートされます。可変長フィールドまたは NULL 可能フィールドがデータの D レコードの最後のフィールドである場合には、そのフィールドの最大長の全体を PC/IXF ファイルに書き込む必要はありません。

FORCEIN オプション:

forcein ファイル・タイプ修飾子を使用すると、PC/IXF ファイルとターゲット・データベースとでデータのコード・ページが違っていても、PC/IXF ファイルをインポートすることができます。このオプションにより、互換性のある列を定義する上でさらに柔軟性が高くなります。

forcein の一般的なセマンティクス

SBCS または DBCS 環境で forcein ファイル・タイプ修飾子を使用する場合、下記の一般的なセマンティクスが適用されます。

- forcein ファイル・タイプ修飾子を使用する際には、注意が必要です。普通は、このオプションを使用可能にせずにインポートを試みるようにしてください。しかし、PC/IXF データ交換アーキテクチャーの一般的な性質のために、一部の PC/IXF ファイルには、介入なしではインポートできないデータ・タイプまたは値が収められている可能性があります。

- forcein を使用して*新しい* 表へのインポートを実行する場合、既存の表へのインポートとは異なる結果が生じる可能性があります。既存の表には、PC/IXF の各データ・タイプごとに事前定義のターゲット・データ・タイプが対応しています。
- lobsinfile ファイル・タイプ修飾子を使用して LOB データがエクスポートされている場合に、ファイルがコード・ページの異なる別のクライアントに移動されると、それぞれ別個のファイル内の CLOB および DBCLOB は、その他のデータの場合とは異なり、データベースへのインポートまたはロード時にクライアントのコード・ページに変換されません。

forcein のコード・ページのセマンティクス

SBCS または DBCS 環境で forcein ファイル・タイプ修飾子を使用する場合、下記のコード・ページのセマンティクスが適用されます。

- forcein ファイル・タイプ修飾子を指定すると、インポート・ユーティリティーのすべてのコード・ページ比較が使用できなくなります。

  この規則は、新しいデータベース表または既存のデータベース表へのインポート時に、列レベルとファイル・レベルのコード・ページ比較に適用されます。列 (例えばデータ・タイプ) レベルでは、この規則は次のデータベース・マネージャーおよび PC/IXF データ・タイプにのみ適用されます。つまり、文字 (CHAR、VARCHAR、および LONG VARCHAR) と GRAPHIC (GRAPHIC、VARGRAPHIC、および LONG VARGRAPHIC)。この制限は、その他のデータ・タイプのコード・ページ属性がデータ・タイプ値の解釈に関係しないという事実に基づいています。
- forcein を指定しても、データ・タイプを判別するためのコード・ページ属性の検査は使用不可にされません。

  例えば、データベース・マネージャーでは、FOR BIT DATA 属性を使用して CHAR 列を宣言することができます。このような宣言により、その列の SBCS CPGID と DBCS CPGID の両方が 0 に設定されます。それらの CPGID の値が 0 の場合、列値は文字ストリングではなくビット・ストリングとして識別されます。
- forcein は、コード・ページ変換を暗黙指定しません。

  forcein ファイル・タイプ修飾子の影響を受けるデータ・タイプの値は、「元のまま」コピーされます。コード・ページ環境の変更に対応するためのコード・ポイント・マッピングは使用されません。ターゲット列が固定長の場合には、インポートされた値にスペースを埋めることが必要になることがあります。
- forcein を使用して*既存の* 表にデータをインポートする場合には、
  - ターゲット・データベース表および列のコード・ページ値が常に優先されます。
  - PC/IXF ファイルおよび列のコード・ページ値は無視されます。

  この規則は、forcein が使用されるかどうかに関係なく適用されます。データベース・マネージャーは、データベースの作成後にデータベースまたは列のコード・ページ値の変更を許可しません。
- forcein を使用して*新しい* 表へのインポートを実行する場合には、
  - ターゲット・データベースのコード・ページ値が優先されます。

– IXFCSBCP = IXFCDBCP = 0 の文字タイプの PC/IXF 列は、FOR BIT DATA としてマークされた表の列を生成します。
– その他のすべての PC/IXF 文字タイプ列は、 SBCS および DBCS CPGID 値がデータベースと等しい文字タイプの表の列を生成します。
– PC/IXF GRAPHIC 列は、SBCS CPGID が「未定義」で、DBCS CPGID がデータベースと等しい表 GRAPHIC 列を生成します (DBCS 環境のみ)。

IXFCSBCP = '00897' および IXFCDBCP = '00301' である PC/IXF CHAR 列を考察してみます。この列を、SBCS CPGID = '00850' および DBCS CPGID = '00000' のデータベース CHAR 列にインポートするとします。 forcein を指定しない場合、ユーティリティーは終了し、データがインポートされないか、あるいは PC/IXF 列値が無視され、データベース列に NULL が入れられます (データベース列が NULL 可能である場合)。 forcein を指定した場合、ユーティリティーは、コード・ページの非互換性を無視して処理を継続します。データ・タイプにその他 (長さなど) の非互換性がなければ、PC/IXF 列の値は「元のまま」インポートされ、データベース列のコード・ページ環境で解釈できるようになります。

以下の 2 つの表には次の点が示されています。

- PC/IXF ファイルのうち指定されたコード・ページ属性を持つデータ・タイプがインポートされる場合に、*新しい* データベース表で作成される列のコード・ページ。
- PC/IXF データ・タイプが無効であるか、または互換性がない場合に、それらがインポート・ユーティリティーによってリジェクトされること。

表 28. インポート・ユーティリティーのコード・ページに関するセマンティクスのサマリー (新しい表) (SBCS 用)

PC/IXF データ・タイプのコード・ページ属性	データベース表の列のコード・ページ属性	
	forcein なし	forcein あり
(0,0)	(0,0)	(0,0)
(a,0)	(a,0)	(a,0)
(x,0)	リジェクト	(a,0)
(x,y)	リジェクト	(a,0)
(a,y)	リジェクト	(a,0)
(0,y)	リジェクト	(0,0)

注:

1. この表では、a と x の間の変換表がないことを想定しています。もしそれがある場合、項目 3 および 4 は、forcein が使用されなくても正常に機能します。
2. 表 29 の注を参照してください。

表 29. インポート・ユーティリティーのコード・ページに関するセマンティクスのサマリー (新しい表) (DBCS 用)

PC/IXF データ・タイプのコード・ページ属性	データベース表の列のコード・ページ属性	
	forcein なし	forcein あり
(0,0)	(0,0)	(0,0)
(a,0)	(a,b)	(a,b)

表 29. インポート・ユーティリティーのコード・ページに関するセマンティクスのサマリー (新しい表) (DBCS 用) (続き)

PC/IXF データ・タイプのコード・ページ属性	データベース表の列のコード・ページ属性	
	forcein なし	forcein あり
(x,0)	リジェクト	(a,b)
(a,b)	(a,b)	(a,b)
(x,y)	リジェクト	(a,b)
(a,y)	リジェクト	(a,b)
(x,b)	リジェクト	(a,b)
(0,b)	(-,b)	(-,b)
(0,y)	リジェクト	(-,b)

注:

1. この表では、a と x の間の変換表がないことを想定しています。
2. PC/IXF データ・タイプのコード・ページ属性は順序対として示されています。x はゼロ以外の 1 バイト・コード・ページ値、y はゼロ以外の 2 バイト・コード・ページ値を表します。 '-' は未定義のコード・ページ値を表します。
3. 各種のコード・ページ属性で、あえて異なる文字を使用しています。異なる文字は値が異なることを暗示しています。例えば、PC/IXF データ・タイプが (x,y) として示されており、データベース列が (a,y) として示されている場合、x は a と等しくありませんが、 PC/IXF ファイルとデータベースの 2 バイト・コード・ページ値は同じ y です。
4. forcein のコード・ページのセマンティクスによって影響を受けるのは、文字および GRAPHIC データ・タイプだけです。
5. 新しい表の入ったデータベースのコード・ページ属性は (a,0) であることが想定されています。このため、新しい表のうち文字タイプのすべての列のコード・ページ属性は (0,0) または (a,0) でなければなりません。

   DBCS 環境において、新しい表を収めたデータベースのコード・ページ属性は (a,b) であることが想定されています。そのため、新しい表のうちすべての GRAPHIC 列のコード・ページ属性は (-,b) でなければならず、文字タイプのすべての列のコード・ページは (a,b) でなければなりません。 SBCS CPGID は、GRAPHIC データ・タイプの場合は未定義であるため、'-' と示されています。
6. 結果のデータ・タイプは、『forcein のデータ・タイプのセマンティクス』で説明されている規則によって決まります。
7. リジェクトの結果は、無効なまたは互換性のないデータ・タイプに関する規則を反映したものです。

以下の 2 つの表には次の点が示されています。

- インポート・ユーティリティーが、さまざまなコード・ページ属性を持つ PC/IXF データ・タイプを、指定されたコード・ページ属性を持つ*既存の* 表の列 (ターゲット 列) に受け入れること。
- インポート・ユーティリティーが、特定のコード・ページ属性を持つ PC/IXF データ・タイプを、指定されたコード・ページ属性を持つ*既存の* 表の列にインポー

トするのを許可しないこと。ユーティリティーは、PC/IXF データ・タイプが無効であるか、または互換性がない場合、それらをリジェクトします。

表 30. インポート・ユーティリティーのコード・ページに関するセマンティクスのサマリー (既存の表) (SBCS 用)

PC/IXF データ・タイプのコード・ページ属性	ターゲット・データベース列のコード・ページ属性	インポートの結果	
		forcein なし	forcein あり
(0,0)	(0,0)	受け入れ	受け入れ
(a,0)	(0,0)	受け入れ	受け入れ
(x,0)	(0,0)	受け入れ	受け入れ
(x,y)	(0,0)	受け入れ	受け入れ
(a,y)	(0,0)	受け入れ	受け入れ
(0,y)	(0,0)	受け入れ	受け入れ
(0,0)	(a,0)	NULL またはリジェクト	受け入れ
(a,0)	(a,0)	受け入れ	受け入れ
(x,0)	(a,0)	NULL またはリジェクト	受け入れ
(x,y)	(a,0)	NULL またはリジェクト	受け入れ
(a,y)	(a,0)	NULL またはリジェクト	受け入れ
(0,y)	(a,0)	NULL またはリジェクト	NULL またはリジェクト

注:

- この表では、a と x の間の変換表がないことを想定しています。
- 283 ページの表 28 の注を参照してください。
- NULL またはリジェクトの結果は、無効なまたは互換性のないデータ・タイプに関する規則を反映したものです。

表 31. インポート・ユーティリティーのコード・ページに関するセマンティクスのサマリー (既存の表) (DBCS 用)

PC/IXF データ・タイプのコード・ページ属性	ターゲット・データベース列のコード・ページ属性	インポートの結果	
		forcein なし	forcein あり
(0,0)	(0,0)	受け入れ	受け入れ
(a,0)	(0,0)	受け入れ	受け入れ
(x,0)	(0,0)	受け入れ	受け入れ
(a,b)	(0,0)	受け入れ	受け入れ
(x,y)	(0,0)	受け入れ	受け入れ
(a,y)	(0,0)	受け入れ	受け入れ
(x,b)	(0,0)	受け入れ	受け入れ
(0,b)	(0,0)	受け入れ	受け入れ

表 31. インポート・ユーティリティーのコード・ページに関するセマンティクスのサマリー (既存の表) (DBCS 用) (続き)

PC/IXF データ・タイプのコード・ページ属性	ターゲット・データベース列のコード・ページ属性	インポートの結果	
		forcein なし	forcein あり
(0,y)	(0,0)	受け入れ	受け入れ
(0,0)	(a,b)	NULL またはリジェクト	受け入れ
(a,0)	(a,b)	受け入れ	受け入れ
(x,0)	(a,b)	NULL またはリジェクト	受け入れ
(a,b)	(a,b)	受け入れ	受け入れ
(x,y)	(a,b)	NULL またはリジェクト	受け入れ
(a,y)	(a,b)	NULL またはリジェクト	受け入れ
(x,b)	(a,b)	NULL またはリジェクト	受け入れ
(0,b)	(a,b)	NULL またはリジェクト	NULL またはリジェクト
(0,y)	(a,b)	NULL またはリジェクト	NULL またはリジェクト
(0,0)	(-,b)	NULL またはリジェクト	受け入れ
(a,0)	(-,b)	NULL またはリジェクト	NULL またはリジェクト
(x,0)	(-,b)	NULL またはリジェクト	NULL またはリジェクト
(a,b)	(-,b)	NULL またはリジェクト	NULL またはリジェクト
(x,y)	(-,b)	NULL またはリジェクト	NULL またはリジェクト
(a,y)	(-,b)	NULL またはリジェクト	NULL またはリジェクト
(x,b)	(-,b)	NULL またはリジェクト	NULL またはリジェクト
(0,b)	(-,b)	受け入れ	受け入れ
(0,y)	(-,b)	NULL またはリジェクト	受け入れ

注:

- この表では、a と x の間の変換表がないことを想定しています。
- 283 ページの表 28 の注を参照してください。
- NULL またはリジェクトの結果は、無効なまたは互換性のないデータ・タイプに関する規則を反映したものです。

forcein のデータ・タイプのセマンティクス

forcein ファイル・タイプ修飾子を使用すると、特定の PC/IXF 列を、データ・タイプが違う、またはその他の点で互換性がないデータ・タイプのターゲット・データベース列にインポートすることができます。SBCS または DBCS 環境で forcein を使用する場合、下記のデータ・タイプのセマンティクスが適用されます (一部の例外を除く)。

- SBCS 環境では、forcein により、次のインポートが可能になります。
  - PC/IXF BIT データ・タイプ (PC/IXF の文字タイプの列で IXFCSBCP = 0 = IXFCDBCP) を、文字タイプのデータベース列 (SBCS CPGID がゼロ以外、DBCS CPGID = 0) にインポートすること。既存の表のみ。
  - PC/IXF MIXED データ・タイプ (IXFCSBCP および IXFCDBCP がゼロ以外) を文字タイプのデータベース列にインポートすること。新しい表と既存の表の両方。
  - PC/IXF GRAPHIC データ・タイプを、データベース FOR BIT DATA 列 (SBCS CPGID = 0 = DBCS CPGID) にインポートすること。新しい表のみ (既存の表では常に可能)。
- forcein ファイル・タイプ修飾子は、有効な PC/IXF データ・タイプの範囲を拡張しません。

  有効な PC/IXF データ・タイプとして定義されていないデータ・タイプの PC/IXF 列は、 forcein が指定されているかどうかにかかわりなく、インポートでは無効です。
- DBCS 環境では、forcein により、次のインポートが可能になります。
  - PC/IXF BIT データ・タイプを文字タイプのデータベース列にインポートすること。
  - PC/IXF BIT データ・タイプをデータベース GRAPHIC 列にインポートすること。ただし、PC/IXF BIT 列が固定長の場合、長さは偶数でなければなりません。長さが奇数である固定長の PC/IXF BIT 列は、データベース GRAPHIC 列と互換性がありません。可変長の PC/IXF BIT 列は、長さが奇数であっても偶数であっても、互換性があります。ただし、可変長列の奇数の長さの値は、データベース GRAPHIC 列へのインポートでは無効です。
  - PC/IXF MIXED データ・タイプを文字タイプのデータベース列にインポートすること。

表 32 は、forcein を指定して PC/IXF ファイルを新しいデータベース表または既存のデータベース表にインポートする操作をまとめたものです。

表 32. forcein を使用する場合の PC/IXF ファイルのインポートのサマリー

PC/IXF の列データ・タイプ	データベースの列データ・タイプ											
	SMALL INT	INT	BIGINT	DEC	FLT	(0,0)	(SBCS, 0)[e]	(SBCS, DBCS)[b]	GRAPH[b]	DATE	TIME	TIME STAMP
-SMALLINT	N											
	E	E	E	E[a]	E							
-INTEGER		N										
	E[a]	E	E	E[a]	E							

表 32. forcein を使用する場合の PC/IXF ファイルのインポートのサマリー (続き)

PC/IXF の列データ・タイプ	データベースの列データ・タイプ											
	SMALL INT	INT	BIGINT	DEC	FLT	(0,0)	(SBCS, 0)[e]	(SBCS, DBCS)[b]	GRAPH[b]	DATE	TIME	TIME STAMP
-BIGINT			N									
	E[a]	E[a]	E	E[a]	E							
-DECIMAL				N								
	E[a]	E[a]	E[a]	E[a]	E							
-FLOAT					N							
	E[a]	E[a]	E[a]	E[a]	E							
-(0,0)												
						N						
-(SBCS,0)						E	E w/F	E w/F	E w/F	E[c]	E[c]	E[c]
							N	N				
-(SBCS, DBCS)						E	E	E		E[c]	E[c]	E[c]
							N w/F[d]	N		E[c]	E[c]	E[c]
						E	E w/F	E				
-GRAPHIC						N w/F[d]			N			
						E			E			
-DATE										N		
										E		
-TIME											N	
											E	
-TIME STAMP												N
												E

[a] ターゲットの数値データ・タイプの範囲内にない値は、その特定の値に関してリジェクトされます。
[b] データ・タイプは DBCS 環境でのみ使用可能です。
[c] 有効な日付または時刻の値でない値は、その特定の値に関してリジェクトされます。
[d] ソース PC/IXF データ・タイプがターゲット・データベースによってサポートされない場合にのみ適用されます。
[e] データ・タイプは DBCS 環境では使用不可です。

注: forcein を使用しなければ PC/IXF 列をデータベース列にインポートできない場合は、'N' または 'E' に 'w/F' を付けて示しています。 'N' は、ユーティリティーが新しいデータベース表を作成することを示します。'E' は、ユーティリティーが既存のデータベース表にデータをインポートすることを示します。 forcein ファイル・タイプ修飾子は、文字および GRAPHIC データ・タイプの互換性にのみ影響を与えます。

データの移動に関する Unicode のための考慮事項

エクスポート、インポート、およびロード・ユーティリティーは、非 Unicode データベースに接続されている Unicode クライアントで使用される場合にはサポートされません。

この項で説明されているとおり、 Unicode データベースには DEL、ASC、および PC/IXF ファイル・フォーマットがサポートされます。

Unicode データベースから ASCII 区切り文字付き (DEL) ファイルにエクスポートする際に、すべての文字データはアプリケーション・コード・ページに変換されます。文字ストリングと GRAPHIC ストリング・データの両方が、クライアントの同じ SBCS または MBCS コード・ページに変換されます。区切り文字付き ASCII ファイル全体でコード・ページは 1 つしかないため、これがどのデータベースのエクスポートの場合でも所定の動作となり、変更できません。したがって、区切り文字付き ASCII ファイルにエクスポートする場合には、ご使用のアプリケーション・コード・ページに存在する UCS-2 文字だけが保管されます。その他の文字は、アプリケーション・コード・ページのデフォルト置換文字に置き換えられます。 UTF-8 クライアント (コード・ページ 1208) の場合、 UTF-8 クライアントによりすべての UCS-2 文字がサポートされているため、データの脱落はありません。

ASCII ファイル (DEL または ASC) から Unicode データベースにインポートする場合、文字ストリング・データはアプリケーション・コード・ページから UTF-8 に変換され、 GRAPHIC ストリング・データはアプリケーション・コード・ページから UCS-2 に変換されます。データの脱落はありません。異なるコード・ページに保管された ASCII データをインポートする場合には、 IMPORT コマンドを発行する前にデータ・ファイル・コード・ページを変更しなければなりません。 **DB2CODEPAGE** レジストリー変数を ASCII データ・ファイルのコード・ページに設定するか、または `codepage` ファイル・タイプ修飾子を使用することにより、データ・ファイルのコード・ページを指定できます。

SBCS および MBCS クライアントで有効な ASCII 区切り文字の範囲は、これらのクライアント用に DB2 for Linux, UNIX, and Windows が現在サポートしているものと同じです。UTF-8 クライアントの有効な区切り文字の範囲は X'01' から X'7F' で、通常の制約事項が適用されます。

Unicode データベースから PC/IXF ファイルにエクスポートする際には、文字ストリング・データはクライアントの SBCS/MBCS コード・ページに変換されます。 GRAPHIC ストリング・データは変換されず、 UCS-2 (コード・ページ 1200) に保管されます。データの脱落はありません。

PC/IXF ファイルから Unicode データベースにインポートする際には、文字ストリング・データは PC/IXF ヘッダーに格納される SBCS/MBCS コード・ページに、 GRAPHIC ストリング・データは PC/IXF ヘッダーに格納される DBCS コード・ページにあるものと見なされます。文字ストリング・データは、インポート・ユーティリティーにより、 PC/IXF ヘッダーで指定されるコード・ページからクライアントのコード・ページに変換され、その後 (INSERT ステートメントによって) クライアント・コード・ページから UTF-8 に変換されます。 GRAPHIC ストリング・データは、インポート・ユーティリティーによって、 PC/IXF ヘッダーで指定される DBCS コード・ページから UCS-2 (コード・ページ 1200) に直接変換されます。

ロード・ユーティリティーは、データをデータベースに直接入れ、デフォルトでは、ASC または DEL ファイル内のデータは、データベースのコード・ページにあるものと見なします。したがって、デフォルトでは、ASCII ファイルのコード・ページ変換は実行されません。データ・ファイルのコード・ページが (`codepage` ファイル・タイプ修飾子を使用して) 明示的に指定された場合、ロード・ユーティリティーは、データをロードする前にこの情報を使用して、指定されるコード・ページからデータベース・コード・ページに変換します。 PC/IXF ファイルでは、ロー

ド・ユーティリティーは、常に IXF ヘッダーに指定されるコード・ページからデータベース・コード・ページ (CHAR の場合は 1208、 GRAPHIC の場合は 1200) に変換します。

DBCLOB ファイルのコード・ページは常に UCS-2 では 1200 です。 CLOB ファイルのコード・ページは、インポート、ロード、あるいはエクスポートされるデータ・ファイルのコード・ページと同じです。例えば、PC/IXF フォーマットを使用してデータをロードまたはインポートする際には、 CLOB ファイルは、PC/IXF ヘッダーによって指定されるコード・ページにあるものと見なされます。 DBCLOB ファイルが ASC または DEL フォーマットの場合には、ロード・ユーティリティーは、CLOB データがデータベースのコード・ページにあるものと見なし、インポート・ユーティリティーは、これがクライアント・アプリケーションのコード・ページにあるものと見なします。

以下の理由で、Unicode データベースには `nochecklengths` 修飾子が常に指定されます。

- DBCS コード・ページのないデータベースには任意の SBCS を接続できます。
- UTF-8 フォーマットの文字ストリングは通常、クライアント・コード・ページにあるものとは異なる長さになります。

コード・ページ 1394、1392、および 5488 に関する考慮事項

インポート、エクスポート、およびロード・ユーティリティーを使用して、中国語コード・ページ GB 18030 (コード・ページ ID 1392 および 5488) および日本語コード・ページ Shift JIS X0213 (コード・ページ ID 1394) から DB2 Unicode データベースにデータを転送できます。加えて、エクスポート・ユーティリティーを使用して、DB2 Unicode データベースから GB 18030 または Shift JIS X0213 コード・ページ・データにデータを転送することもできます。

例えば、以下のコマンドは、リモート接続クライアント上にある Shift JIS X0213 データ・ファイル `u/jp/user/x0213/data.del` を MYTABLE にロードします。

```
db2 load client from /u/jp/user/x0213/data.del
of del modified by codepage=1394 insert into mytable
```

ここで、MYTABLE は DB2 Unicode データベース内にあります。

Unicode クライアントと Unicode サーバーとの間の接続しかサポートされないため、 Unicode クライアントを使用するか、あるいはロード、インポート、またはエクスポート・ユーティリティーを使用する前に、 DB2 レジストリー変数 **DB2CODEPAGE** を 1208 に設定する必要があります。

コード・ページ 1394 から Unicode に変換すると、拡張が行われます。例えば、2 バイト文字は、GRAPHIC 列で 2 つの 16 ビット Unicode 文字として格納されます。 Unicode データベースのターゲット列は、拡張された Unicode バイトを入れるのに十分な幅であることを確認する必要があります。

非互換性

Unicode データベースに接続されているアプリケーションの場合、 GRAPHIC ストリング・データは常に UCS-2 (コード・ページ 1200) にあります。非 Unicode データベースに接続されるアプリケーションの場合、 GRAPHIC ストリング・データ

は、アプリケーションの DBCS コード・ページにあるか、またはアプリケーション・コード・ページが SBCS の場合には許可されません。例えば、日本語の非 Unicode データベースに 932 クライアントが接続される場合、 GRAPHIC ストリング・データはコード・ページ 301 に入れられます。 Unicode データベースに接続されている 932 クライアント・アプリケーションの場合、 GRAPHIC ストリング・データは UCS-2 エンコード方式になります。

文字セットと各国語サポート

DB2 データ移動ユーティリティーには、次のような各国語サポートが備わっています。

- インポートおよびエクスポート・ユーティリティーには、クライアント・コード・ページからサーバー・コード・ページへの自動コード・ページ変換が備わっています。
- ロード・ユーティリティーでは、 `codepage` 修飾子とともに DEL および ASC ファイルを指定して、任意のコード・ページからサーバー・コード・ページにデータを変換することができます。
- どのユーティリティーでも、IXF データは、元のコード・ページ (IXF ファイルに保管されているもの) からサーバーのコード・ページに自動的に変換されます。

場合によって、コード・ページが異なるために文字データの長さに変化が生じることがあります。例えば、日本語または中国語 (繁体字) の拡張 UNIX コード (EUC) と 2 バイト文字セット (DBCS) では、同じ文字が別々の長さでエンコードされていることがあります。普通、入力データの長さとターゲット列の長さの比較は、まだどのデータも読み取らないうちに実行されます。入力の長さがターゲットの長さを超えている場合、列が NULL 可能であれば NULL がその列に挿入されます。そうでない場合、要求はリジェクトされます。 `nochecklengths` ファイル・タイプ修飾子を指定した場合は、初期の比較を実行しないで、データのインポートまたはロードを実行しようとします。変換完了後にデータが長すぎることが明らかになった場合、その行はリジェクトされます。そうでなければ、データはインポートされるかロードされます。

XML データ移動

XML データ移動のサポートは、ロード、インポート、およびエクスポート・ユーティリティーによって提供されます。 ADMIN_MOVE_TABLE ストアード・プロシージャーを使用すると、表をオフラインにしないでも XML 列を含む表を移動することができます。

XML データのインポート

インポート・ユーティリティーを使用して、 XML 文書を通常のリレーショナル表に挿入できます。インポートできるのは、整形式 XML 文書だけです。

IMPORT コマンドの XML FROM オプションを使用して、インポートする XML 文書の場所を指定します。 XMLVALIDATE オプションは、インポートされた文書を妥当性検査する方法を指定します。インポートされた XML データが、IMPORT コマンドで指定されたスキーマに対して、ソース XML 文書内のスキーマ・ロケーション・ヒントによって識別されるスキーマに対して、またはメイン・データ・フ

ファイル内の XML Data Specifier によって識別されるスキーマに対して、妥当性検査されるように選択できます。また、XMLPARSE オプションを使用して、XML 文書がインポートされるときの空白文字の処理方法を指定できます。 xmlchar および xmlgraphic ファイル・タイプ修飾子によって、インポートされる XML データのエンコード特性を指定できます。

XML データのロード

ロード・ユーティリティーは、大量の XML データを表に挿入するための効果的な方法を提供します。このユーティリティーはまた、ユーザー定義のカーソルからロードする機能など、インポート・ユーティリティーでは使用できない特定のオプションを使用することができます。

IMPORT コマンドと同じように LOAD コマンドでも、ロードする XML データの場所、XML データの妥当性検査オプション、および空白文字の処理方法を指定できます。 IMPORT と同様に、xmlchar および xmlgraphic ファイル・タイプ修飾子を使用して、ロードされた XML データのエンコード特性を指定できます。

XML データのエクスポート

データは、XML データ・タイプの列を 1 列以上含む表からエクスポートできます。エクスポートされた XML データは、エクスポートされたリレーショナル・データを含むメイン・データ・ファイルとは別個のファイルに格納されます。各エクスポート XML 文書についての情報は、エクスポートされたメインのデータ・ファイル内で XML データ指定子 (XDS) によって表されます。 XDS は、XML 文書が保管されるシステム・ファイルの名前、このファイル内の XML 文書の正確な場所と長さ、および XML 文書の妥当性検査に使用される XML スキーマを指定するストリングです。

EXPORT コマンドの XMLFILE、XML TO、および XMLSAVESCHEMA パラメーターを使用することにより、エクスポートされた XML 文書の格納方法についての詳細を指定できます。 xmlinsepfiles、xmlnodeclaration、xmlchar、および xmlgraphic ファイル・タイプ修飾子により、エクスポートされた XML データの保管場所およびエンコード方式に関する詳細を指定できます。

オンラインでの表の移動

ADMIN_MOVE_TABLE ストアード・プロシージャーは、データをオンラインでアクセス可能な状態のままで、そのアクティブ表のデータを同じ名前の新しい表オブジェクトに移動できます。表には、XML データ・タイプの 1 つ以上の列を含めることができます。コスト、スペース、移動のパフォーマンス、トランザクション・オーバーヘッドよりも可用性の方を重視する場合には、オフラインの表移動ではなく、オンラインの表移動を使用してください。

このプロシージャーは、1 回呼び出すことも、複数回 (プロシージャーが実行する各操作につき 1 回ずつ) 呼び出すこともできます。複数の呼び出しを行う場合、移動のキャンセルや、ターゲット表が更新のためにオフラインになるタイミングの制御などの追加オプションを使用できます。

XML データの移動に関する重要な考慮事項

XML データをインポートまたはエクスポートする際には、考慮すべきいくつかの制限事項、前提条件、および注意事項があります。XML データをインポートまたはエクスポートする前に、こうした考慮事項を検討してください。

XML データをエクスポートまたはインポートする際に、以下の考慮事項に注意を払ってください。

- エクスポートされた XML データは、エクスポートされたリレーショナル・データを含むメイン・データ・ファイルとは別個の場所に常に保管されます。
- デフォルトで、エクスポート・ユーティリティーは XML データを Unicode で書き込みます。 `xmlchar` ファイル・タイプ修飾子を使用して、XML データを文字コード・ページで書き込むようにするか、`xmlgraphic` ファイル・タイプ修飾子を使用して、アプリケーションのコード・ページに関わりなく XML データを UTF-16 (グラフィック・コード・ページ) で書き込むようにします。
- XML データは Unicode 以外のデータベースで格納でき、XML 列に挿入されるデータは挿入前にデータベース・コード・ページから UTF-8 に変換されます。XML 構文解析中に代替文字が使用されないようにするため、挿入する文字データはデータベース・コード・ページに含まれるコード・ポイントのみを使用して構成する必要があります。 `enable_xmlchar` 構成パラメーターを `no` に設定すると、XML 構文解析中に文字データ・タイプの挿入がブロックされて、BIT DATA、BLOB や XML など、コード・ページ変換を実施しないデータ・タイプの挿入が制限されます。
- XML データをインポートまたはロードする際、インポートする XML 文書にエンコード属性を含む宣言タグが含まれていない限り、XML データは Unicode であると想定されます。`xmlchar` ファイル・タイプ修飾子を使用して、インポートする XML 文書が文字コード・ページでエンコードされるように指定できます。一方、`xmlgraphic` ファイル・タイプ修飾子は、インポートする XML 文書が UTF-16 でエンコードされることを指定します。
- インポート・ユーティリティーとロード・ユーティリティーは、整形式でない文書を含む行を拒否します。
- `XMLVALIDATE` オプションがインポート・ユーティリティーまたはロード・ユーティリティーに指定されている場合、マッチング・スキーマに対する妥当性検査が成功した文書は、表に挿入されるときに、妥当性検査に使用されたスキーマに関する情報のアノテーションが付けられます。マッチング・スキーマに対する妥当性検査が失敗した文書を含む行は、拒否されます。
- `XMLVALIDATE` オプションがインポート・ユーティリティーまたはロード・ユーティリティーに指定され、複数の XML スキーマを使用して XML 文書を妥当性検査する場合、カタログ・キャッシュ・サイズ構成パラメーター **`catalogcache_sz`** を増やす必要がある場合があります。**`catalogcache_sz`** の値を増やすことが適切でないか不可能な場合、単一のインポート・コマンドまたはロード・コマンドを、もっと少ないスキーマ文書しか使用しない複数のコマンドに分けることができます。
- XQuery ステートメントを指定して XML データをエクスポートする場合、整形式 XML 文書ではない Query および XPath データ・モデル (XDM) のインスタンスをエクスポートできます。 XML データ・タイプで定義された列には完全な

整形式の XML 文書だけしか含めることができないので、エクスポートされた整形式ではない XML 文書は XML 列に直接インポートできません。

- ロード中の **CPU_PARALLELISM** 設定は、統計が収集されている場合、1 に縮小されます。
- XML ロード操作を続行するには、共有ソート・メモリーを使用することが必要です。 **SHEAPTHRES_SHR** または **INTRA_PARALLEL** を有効にするか、または接続コンセントレーターをオンにします。デフォルトでは **SHEAPTHRES_SHR** は設定されているので、共有ソート・メモリーはデフォルト構成で使用できます。
- XML 列を含む表をロードする場合、LOAD コマンドの **SOURCEUSEREXIT** オプションまたは **SAVECOUNT** パラメーターを指定することはできません。
- LOAD コマンドを使用する際、LOB ファイルの場合と同様に、XML ファイルもサーバー・サイドに存在する必要があります。
- XML データをパーティション・データベース環境にある複数のデータベース・パーティションにロードする場合、XML データを含むファイルはすべてのデータベース・パーティションにアクセス可能でなければなりません。例えば、ファイルをコピーするか、NFS マウントを作成して、ファイルをアクセス可能にすることができます。

インポートおよびエクスポート時の LOB および XML ファイルの振る舞い

LOB および XML ファイルは、データをインポートおよびエクスポートする時に使用できる特定の性質および互換性を共有します。

エクスポート

データをエクスポートするときに LOBS TO オプションを付けて 1 つ以上の LOB パスを指定する場合、エクスポート・ユーティリティーはパスの間を循環し、それぞれの連続した LOB 値を適切な LOB ファイルに書き込みます。同様に、1 つ以上の XML パスが XML TO オプションで指定された場合、エクスポート・ユーティリティーはそれらのパスの間で循環して、それぞれの連続した XQuery および XPath データ・モデル (XDM) インスタンスを該当する XML ファイルに書き込みます。デフォルトでは、LOB 値および XDM インスタンスは、エクスポートされるリレーショナル・データが書き込まれているパスと同じパスに書き込まれます。
LOBSINSEPFILES または XMLINSEPFILES ファイル・タイプ修飾子が設定されているのでないかぎり、LOB ファイルと XML ファイルは両方とも複数の値を同じファイルに連結できます。

LOBFILE オプションを指定すると、エクスポート・ユーティリティーによって生成される LOB ファイルのベース名を指定することができます。同様に、XMLFILE オプションを指定すると、エクスポート・ユーティリティーによって生成される XML ファイルのベース名を指定することができます。エクスポート・データ・ファイルの名前に拡張子 .lob を付けたものが、デフォルトの LOB ファイルのベース名になります。エクスポート・データ・ファイルの名前に拡張子 .xml を付けたものが、デフォルトの XML ファイルのベース名になります。したがって、エクスポート LOB ファイルまたは XML ファイルの完全名は、ベース名、3 桁の数値の拡張子、そして拡張子 .lob または .xml をこの順番でつなぎ合わせたもので構成されます。

インポート

データをインポートするとき、LOB ロケーション指定子 (LLS) には XML ターゲット列との互換性があり、XML データ指定子 (XDS) には LOB ターゲット列との互換性があります。 LOBS FROM オプションが指定されない場合、インポートする LOB ファイルは、入力リレーショナル・データ・ファイルと同じパスにあると見なされます。同様に、XML FROM オプションが指定されない場合、インポートする XML ファイルは、入力リレーショナル・データ・ファイルと同じパスにあると見なされます。

エクスポートの例

以下の例では、LOB 値はすべて /mypath/t1export.del.001.lob ファイルに書き込まれ、XDM インスタンスはすべて /mypath/t1export.del.001.xml ファイルに書き込まれます。

```
EXPORT TO /mypath/t1export.del OF DEL MODIFIED BY LOBSINFILE
SELECT * FROM USER.T1
```

以下の例では、最初の LOB 値は /lob1/t1export.del.001.lob ファイルに書き込まれ、2 番目は /lob2/t1export.del.002.lob ファイルに書き込まれ、3 番目は /lob1/t1export.del.001.lob に付加され、4 番目は /lob2/t1export.del.002.lob に付加され、以降このパターンで付加されていきます。

```
EXPORT TO /mypath/t1export.del OF DEL LOBS TO /lob1,/lob2
MODIFIED BY LOBSINFILE SELECT * FROM USER.T1
```

以下の例では、最初の XDM インスタンスは /xml1/xmlbase.001.xml ファイルに書き込まれ、2 番目は /xml2/xmlbase.002.xml ファイル、3 番目は /xml1/xmlbase.003.xml、4 番目は /xml2/xmlbase.004.xml、というパターンで書き込まれていきます。

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /xml1,/xml2 XMLFILE xmlbase
   MODIFIED BY XMLINSEPFILES SELECT * FROM USER.T1
```

インポートの例

XML 列を 1 つ含む「mytable」という表があり、以下の IMPORT コマンドがあるとします。

```
IMPORT FROM myfile.del of del LOBS FROM /lobpath XML FROM /xmlpath
MODIFIED BY LOBSINFILE XMLCHAR replace into mytable
```

「myfile.del」に以下のデータが含まれるとします。

```
mylobfile.001.lob.123.456/
```

この場合、インポート・ユーティリティーは、/lobpath/mylobfile.001.lob ファイルから、ファイル・オフセットを 123 から開始し、長さ 456 バイトで XML 文書のインポートを試行します。

ファイル「mylobfile.001.lob」は XML パスではなく LOB パスにあると見なされます。値が XML データ指定子 (XDS) ではなく LOB ロケーション指定子 (LLS) によって参照されているからです。

XMLCHAR ファイル・タイプ修飾子が指定されているので、文書は文字コード・ページでエンコードされるものと見なされます。

XML データ指定子

エクスポート、インポート、およびロード・ユーティリティーを使用して移動される XML データは、メイン・データ・ファイルとは別のファイルに格納する必要があります。 XML データは、メイン・データ・ファイル内で XML データ指定子 (XDS) を使用して表されます。

XDS は XDS という名前の XML タグによって表されるストリングであり、列内の実際の XML データについての情報を記述する属性が付随しています。そのような情報には、実際の XML データが含まれているファイルの名前、およびそのファイル内の XML データのオフセットおよび長さが含まれます。 XDS の属性について、以下のリストで説明します。

FIL XML データが入っているファイルの名前。Named PIPE を指定することはできません。Named PIPE からの XML 文書のインポートとロードはサポートされていません。

OFF FIL 属性で名前指定されているファイル中の XML データのバイト・オフセット。このオフセットは 0 から始まります。

LEN FIL 属性で名前指定されているファイル中の XML データのバイト単位の長さ。

SCH この XML 文書の妥当性検査に使用する XML スキーマの完全修飾 SQL ID。この SQL ID のスキーマと名前のコンポーネントは、それぞれこの XML スキーマに対応する SYSCAT.XSROBJECTS カタログ表の行の「OBJECTSCHEMA」および「OBJECTNAME」値として保管されます。

XDS はデータ・ファイル中で文字フィールドとして解釈され、ファイル形式の文字の列に関する構文解析動作の対象になります。例えば、区切り文字付き ASCII ファイル形式 (DEL) の場合、区切り文字が XDS 中にあれば、二重にしなければなりません。属性値の中の特殊文字 <、>、&、'、" は常にエスケープしなければなりません。大文字と小文字の区別のあるオブジェクト名は、" 文字エンティティーで囲まなければなりません。

例

値が `abc&"def".del` の FIL 属性について考えてみましょう。区切り文字付き ASCII ファイルにこの XDS を組み込むには、区切り文字が " 文字であれば、" 文字を二重にして、特殊文字をエスケープします。

```
<XDS FIL=""abc&amp;&quot;def&quot;.del"" />
```

以下の例は、区切り文字付き ASCII データ・ファイル中にある XDS を示しています。XML データはファイル `xmldocs.xml.001` に保管され、バイト・オフセットは 100 で始まり長さは 300 バイトになります。この XDS は二重引用符で区切られる ASCII ファイル中にあるので、XDS タグ自体の中の二重引用符は二重にしなければなりません。

```
"<XDS FIL = ""xmldocs.xml.001"" OFF=""100"" LEN=""300"" />"
```

以下の例は、完全修飾 SQL ID `ANTHONY.purchaseOrderTest` を示しています。XDS 中で、ID の大文字と小文字の区別がある部分は " 文字エンティティーで囲まなければなりません。

```
"<XDS FIL='/home/db2inst1/xmlload/a.xml' OFF='0' LEN='6758'
   SCH='ANTHONY.&quot;purchaseOrderTest&quot;' />"
```

Query および XPath のデータ・モデル

SQL で使用できる XQuery 関数を使用するか、または XQuery を直接呼び出すことにより、表内の XML データにアクセスできます。 Query および XPath データ・モデル (XDM) のインスタンスは、整形式 XML 文書、ノードまたはアトミック値のシーケンス、またはノードとアトミック値の任意の組み合わせとすることができます。

個々の XDM インスタンスは、EXPORT コマンドによって 1 つ以上の XML ファイルに書き込むことができます。

付録 A. インポート・ユーティリティーとロード・ユーティリティーの相違点

下記の表に、DB2 ロード・ユーティリティーとインポート・ユーティリティーの主要な相違点を要約します。

インポート・ユーティリティー	ロード・ユーティリティー
大量のデータを移動する場合、処理速度が遅くなります。	大量のデータを移動する場合、インポート・ユーティリティーより処理が速くなります。ロード・ユーティリティーは、フォーマット設定されたページをデータベースに直接書き込むためです。
パーティション内並列処理の利用が制限されます。パーティション内並列処理は、ALLOW WRITE ACCESS モードでインポート・ユーティリティーを並行して呼び出すことによりのみ、行うことができます。	パーティション内並列処理を利用します。一般にこれには対称マルチプロセッサー (SMP) マシンが必要です。
FASTPARSE はサポートされません。	FASTPARSE サポートによって、ユーザー提供データのデータ・チェックが簡略化されます。
階層データがサポートされます。	階層データはサポートされません。
PC/IXF フォーマットでの表、階層、および索引の作成がサポートされます。	表および索引は存在していなければなりません。
マテリアライズ照会表へのインポートはサポートされません。	マテリアライズ照会表へのロードがサポートされます。
BINARYNUMERICS はサポートされません。	BINARYNUMERICS はサポートされます。
PACKEDDECIMAL はサポートされません。	PACKEDDECIMAL はサポートされます。
ZONEDDECIMAL はサポートされません。	ZONEDDECIMAL はサポートされます。
GENERATED ALWAYS として定義された列をオーバーライドできません。	`generatedoverride` および `identityoverride` ファイル・タイプ修飾子を使用して、 GENERATED ALWAYS 列をオーバーライドできます。
表、ビュー、およびニックネームへのインポートがサポートされます。	表へのロードのみサポートされます。
すべての行がログに記録されます。	最小限のロギングが実行されます。
トリガー・サポートがあります。	トリガー・サポートはありません。
インポート操作が中断された場合に、*commitcount* が指定されていた場合、表は使用可能であり、最後の COMMIT までにロードされた行が収められています。ユーザーはインポート操作を再開するか、表を現状のまま受け入れることができます。	ロード操作が中断された場合に、*savecount* が指定されていると、表はロード・ペンディング状態のままになります。その表は、ロード操作が再開されるか、ロード終了操作が呼び出されるか、またはロード操作の試行前に作成されたバックアップ・イメージから表スペースがリストアされるまで、使用できません。

インポート・ユーティリティー	ロード・ユーティリティー
必要なスペースは、最大の索引のサイズ + 10% とほぼ同じです。このスペースは、データベース内の TEMPORARY 表スペースから取得されます。	必要なスペースは、表に関して定義されているすべての索引のサイズの合計とほぼ同じであり、そのサイズの 2 倍まで大きくなる可能性があります。このスペースは、データベース内の一時スペースから取得されます。
インポート操作中にすべての制約が妥当性検査されます。	ロード・ユーティリティーは、固有性をチェックし、生成される列値を計算するが、他のすべての制約 SET INTEGRITY を使用してチェックしなければなりません。
キー値は、インポート操作中に一度に 1 つずつ索引に挿入されます。	データがロードされた後で、キー値がソートされ、索引が作成されます。
統計情報を更新する必要がある場合は、インポート操作の後で RUNSTATS ユーティリティーを実行しなければなりません。	表内のすべてのデータを置換する場合は、ロード操作中に統計情報を収集できます。
DB2 Connect によってホスト・データベースへのインポートが可能です。	ホスト・データベースへのロードはできません。
インポート・ファイルは、インポート・ユーティリティーが呼び出されるクライアントになければなりません。	指定するオプションに応じて、ロード・ファイルまたはパイプは、データベースを備えたデータベース・パーティション上か、またはロード・ユーティリティーを呼び出すリモート接続クライアント上のどちらかに置くことができます。 **注:** LOB および XML データはサーバー・サイドからのみ読み取ることができます。
バックアップ・イメージは不要です。インポート・ユーティリティーでは SQL の挿入が使用されるため、アクティビティーがログに記録され、障害時にそれらの操作をリカバリーするのにバックアップは不要です。	ロード操作中にバックアップ・イメージを作成することができます。

付録 B. エクスポート・ユーティリティー、インポート・ユーティリティー、ロード・ユーティリティーで使用するバインド・ファイル

以下の表は、バインド・ファイルとそのデフォルト分離レベルをリストし、さらにどのユーティリティーがどんな目的で使用するかを説明しています。

バインド・ファイル (デフォルト分離レベル)	ユーティリティー/目的
db2ueiwi.bnd (CS)	インポート/エクスポート。表の列および索引に関する情報を照会するために使用されます。
db2uexpm.bnd (CS)	エクスポート。エクスポート操作に指定される照会をフェッチするのに使用されます。
db2uimpm.bnd (RS)	インポート。 INSERT、REPLACE または REPLACE_CREATE オプションが使用されたときに、ソース・データ・ファイルからターゲット表にデータを挿入するために使用されます。 **注:** 注: IMPORT コマンドの CREATE オプションと REPLACE_CREATE オプションは非推奨で、将来のリリースでは廃止される可能性があります。
db2uipkg.bnd (CS)	インポート。 BIND オプションをチェックするために使用されます。
db2ucktb.bnd (CS)	ロード。ロード操作で一般初期化プロセスを実行するために使用されます。
db2ulxld.bnd (CS)	ロード。カーソル操作によってロード時に提供される照会を処理するために使用されます。
db2uigsi.bnd (UNIX ベースのシステムでは RS、その他のプラットフォームでは RR)	インポート/エクスポート。索引をドロップし、REPLACE オプションでのインポートで参照制約をチェックするために使用されます。また、 IXF ファイルをエクスポートするための ID 列情報を検索するのにも使用されます。
db2uqtpd.bnd (RR)	インポート/エクスポート。階層表の処理を実行するために使用されます。
db2uimtb.bnd (RS)	インポート。インポート操作で一般初期化プロセスを実行するために使用されます。
db2uImpInsUpdate.bnd (RS)	インポート。 INSERT_UPDATE オプションが使用されたときに、ソース・データ・ファイルからターゲット表にデータを挿入するために使用されます。 INSERT BUF オプションを使用してバインドすることはできません。

バインド・ファイル (デフォルト分離レベル)	ユーティリティー/目的
db2uiDescribe.bnd (RS)	インポート/エクスポート。処理されている表の定義とプロパティーについての情報を照会するために使用されます。例えば、処理されている表に対して SQL DESCRIBE を実行するラッパー・モジュールなどです。

付録 C. 構文図の見方

このトピックでは、SQL 構文ダイアグラムの構造について説明します。

構文図は、左から右、上から下に、線に沿って読みます。

記号 ▶▶── は、構文図の始まりを示します。

記号 ──▶ は、構文が次の行に続くことを示します。

記号 ▶── は、構文が前の行から続いていることを示します。

記号 ──▶◀ は、構文図の終わりを示します。

構文フラグメントは、記号 ├── で始まり、記号 ──┤ で終わります。

必須項目は、横線 (メインパス) 上に示されます。

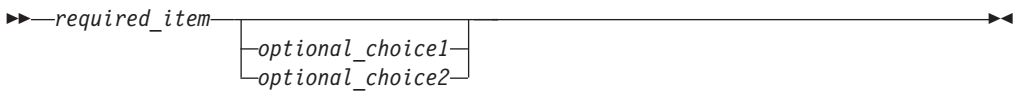
オプショナル項目は、メインパスの下に示されます。

オプショナル項目をメインパスの上に示すこともありますが、それは構文図を見やすくするためであり、実行には関係しません。

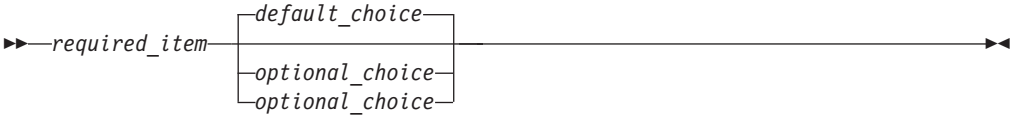
複数の項目からの選択が可能な場合、それらの項目を縦に並べて (スタックに) 示しています。

項目から 1 つを選択しなければならない 場合、スタックの項目の 1 つはメインパス上に示されます。

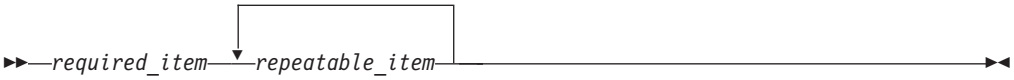
項目から 1 つをオプションで選択できる場合、スタック全体がメインパスよりも下に示されます。



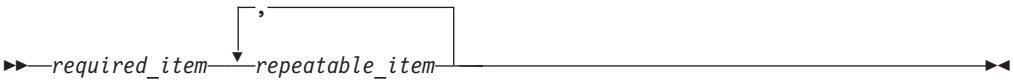
項目の 1 つがデフォルト値の場合、その項目はメインパスより上に示され、残りの選択項目はメインパスよりも下に示されます。



メインパスの上に、左へ戻る矢印がある場合には、項目を繰り返して指定できることを示しています。このような場合、繰り返す項目相互の間は、1 つ以上のブランクで区切らなければなりません。



繰り返しの矢印にコンマが示されている場合は、繰り返し項目をコンマで区切らなければなりません。

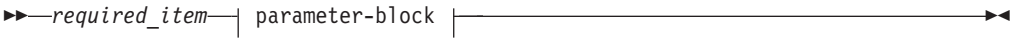


スタックの上部の反復の矢印の記号は、そのスタックの中から複数の項目を選択できること、または 1 つの選択項目を繰り返して選択できることを示します。

キーワードは英大文字で示してあります (例: FROM)。示されているとおりに入力することが必要です。変数は英小文字で示しています (例: column-name)。このような変数は、構文にユーザーが指定する名前や値を示しています。

句読点、括弧、算術演算子、その他の記号が示されている場合には、それらを構文の一部として入力する必要があります。

1 つの変数が、構文を構成する大きいフラグメントを表すことがあります。例えば次の図で、変数 parameter-block は、 **parameter-block** というラベルの構文フラグメント全体を表します。



parameter-block:

黒丸 (●) ではさまれて隣接しているセグメントは、任意の順序で指定することができます。

上記の図は、item2 と item3 をどのような順序で指定しても構わないことを示しています。以下はいずれも有効です。

```
required_item item1 item2 item3 item4
required_item item1 item3 item2 item4
```

付録 D. データ移動問題についてのデータの収集

データ移動コマンドの実行中に問題が発生し、その問題の原因が判別できない場合は、問題を診断して解決するために、あなた自身または IBM ソフトウェア・サポートで使用できる診断データを収集してください。

以下のリストから、発生している事情に該当するデータ収集の指示に従ってください。

- **db2move** コマンドに関連した問題のデータを収集するには、コマンドを発行したディレクトリーに移動してください。コマンドで指定したアクションに応じて、以下のファイルを見つけます。
  - COPY アクションの場合、COPY.*timestamp*.ERR および COPYSCHEMA.*timestamp*.MSG というファイルを探します。さらに LOAD_ONLY または DDL_AND_LOAD モードのいずれかを指定した場合、LOADTABLE.*timestamp*.MSG というファイルも探してください。
  - EXPORT アクションの場合、EXPORT.out というファイルを探してください。
  - IMPORT アクションの場合、IMPORT.out というファイルを探してください。
  - LOAD アクションの場合、LOAD.out というファイルを探してください。
- **EXPORT**、**IMPORT**、または **LOAD** コマンドに関連した問題のデータを収集するには、コマンドに MESSAGES パラメーターが含まれているかどうかを判別します。含まれている場合、出力ファイルを収集します。これらのユーティリティーは、現行ディレクトリーおよびデフォルト・ドライブ以外を指定しない場合にはそれらを宛先として使用します。
- **REDISTRIBUTE** コマンドに関連した問題のデータを収集するには、Linux および UNIX オペレーティング・システムの場合には「*databasename.database_partition_groupname. timestamp*」、Windows オペレーティング・システムの場合には「*databasename. database_partition_groupname.date.time*」というファイルを探してください。それは、Linux および UNIX オペレーティング・システムの場合には *$HOME*/sqllib/db2dump ディレクトリー、Windows オペレーティング・システムの場合には $DB2PATH¥sqllib¥redist ディレクトリーに置かれています。*$HOME* はインスタンス所有者のホーム・ディレクトリーです。

付録 E. DB2 技術情報の概説

DB2 技術情報は、さまざまな方法でアクセスすることが可能な、各種形式で入手できます。

DB2 技術情報は、以下のツールと方法を介して利用できます。

- DB2インフォメーション・センター
  - トピック (タスク、概念、およびリファレンス・トピック)
  - サンプル・プログラム
  - チュートリアル
- DB2 資料
  - PDF ファイル (ダウンロード可能)
  - PDF ファイル (DB2 PDF DVD に含まれる)
  - 印刷資料
- コマンド行ヘルプ
  - コマンド・ヘルプ
  - メッセージ・ヘルプ

注: DB2 インフォメーション・センターのトピックは、PDF やハードコピー資料よりも頻繁に更新されます。最新の情報を入手するには、資料の更新が発行されたときにそれをインストールするか、ibm.com にある DB2 インフォメーション・センターを参照してください。

技術資料、ホワイト・ペーパー、IBM Redbooks® 資料などのその他の DB2 技術情報には、オンライン (ibm.com) でアクセスできます。 DB2 Information Management ソフトウェア・ライブラリー・サイト (http://www.ibm.com/software/data/sw-library/) にアクセスしてください。

資料についてのフィードバック

DB2 の資料についてのお客様からの貴重なご意見をお待ちしています。DB2 の資料を改善するための提案については、db2docs@ca.ibm.com まで E メールを送信してください。DB2 の資料チームは、お客様からのフィードバックすべてに目を通しますが、直接お客様に返答することはありません。お客様が関心をお持ちの内容について、可能な限り具体的な例を提供してください。特定のトピックまたはヘルプ・ファイルについてのフィードバックを提供する場合は、そのトピック・タイトルおよび URL を含めてください。

DB2 お客様サポートに連絡する場合には、この E メール・アドレスを使用しないでください。資料を参照しても、DB2 の技術的な問題が解決しない場合は、お近くの IBM サービス・センターにお問い合わせください。

DB2 テクニカル・ライブラリー (ハードコピーまたは PDF 形式)

以下の表は、IBM Publications Center (www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss) から利用できる DB2 ライブラリーについて説明しています。英語および翻訳された DB2 バージョン 10.1 のマニュアル (PDF 形式) は、www.ibm.com/support/docview.wss?rs=71&uid=swg27009474 からダウンロードできます。

この表には印刷資料が入手可能かどうかが示されていますが、国または地域によっては入手できない場合があります。

資料番号は、資料が更新される度に大きくなります。資料を参照する際は、以下にリストされている最新版であることを確認してください。

注: *DB2 インフォメーション・センター*は、PDF やハードコピー資料よりも頻繁に更新されます。

表 33. DB2 の技術情報

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
管理 API リファレンス	SA88-4671-00	入手可能	2012 年 4 月
管理ルーチンおよびビュー	SA88-4672-01	入手不可	2013 年 1 月
コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻	SA88-4676-01	入手可能	2013 年 1 月
コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻	SA88-4677-01	入手可能	2013 年 1 月
コマンド・リファレンス	SA88-4673-01	入手可能	2013 年 1 月
データベース: 管理の概念および構成リファレンス	SA88-4662-01	入手可能	2013 年 1 月
データ移動ユーティリティー: ガイドおよびリファレンス	SA88-4693-01	入手可能	2013 年 1 月
データベースのモニタリング ガイドおよびリファレンス	SA88-4663-01	入手可能	2013 年 1 月
データ・リカバリーと高可用性 ガイドおよびリファレンス	SA88-4694-01	入手可能	2013 年 1 月
データベース・セキュリティー・ガイド	SA88-4695-01	入手可能	2013 年 1 月

表 33. DB2 の技術情報 (続き)

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
DB2 ワークロード管理ガイドおよびリファレンス	SA88-4685-01	入手可能	2013 年 1 月
ADO.NET および OLE DB アプリケーションの開発	SA88-4665-01	入手可能	2013 年 1 月
組み込み SQL アプリケーションの開発	SA88-4666-01	入手可能	2013 年 1 月
Java アプリケーションの開発	SA88-4669-01	入手可能	2013 年 1 月
Perl、PHP、Python および Ruby on Rails アプリケーションの開発	SA88-4670-00	入手不可	2012 年 4 月
IBM データ・サーバー用の RDF アプリケーション開発	SA88-5083-00	入手可能	2013 年 1 月
SQL および外部ルーチンの開発	SA88-4667-01	入手可能	2013 年 1 月
データベース・アプリケーション開発の基礎	GI88-4279-01	入手可能	2013 年 1 月
DB2 インストールおよび管理 概説 (Linux および Windows 版)	GI88-4280-00	入手可能	2012 年 4 月
グローバリゼーション・ガイド	SA88-4696-00	入手可能	2012 年 4 月
DB2 サーバー機能 インストール	GA88-4679-01	入手可能	2013 年 1 月
IBM データ・サーバー・クライアント機能 インストール	GA88-4680-00	入手不可	2012 年 4 月
メッセージ・リファレンス 第 1 巻	SA88-4688-01	入手不可	2013 年 1 月
メッセージ・リファレンス 第 2 巻	SA88-4689-01	入手不可	2013 年 1 月
Net Search Extender 管理およびユーザーズ・ガイド	SA88-4691-01	入手不可	2013 年 1 月
パーティションおよびクラスタリングのガイド	SA88-4697-01	入手可能	2013 年 1 月
Preparation Guide for DB2 10.1 Fundamentals Exam 610	SC27-4540-00	入手不可	2013 年 1 月

表 33. DB2 の技術情報 (続き)

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
Preparation Guide for DB2 10.1 DBA for Linux, UNIX, and Windows Exam 611	SC27-4541-00	入手不可	2013 年 1 月
pureXML ガイド	SA88-4686-01	入手可能	2013 年 1 月
Spatial Extender ユーザーズ・ガイドおよびリファレンス	SA88-4690-00	入手不可	2012 年 4 月
SQL プロシージャー言語: アプリケーションのイネーブルメントおよびサポート	SA88-4668-01	入手可能	2013 年 1 月
SQL リファレンス 第 1 巻	SA88-4674-01	入手可能	2013 年 1 月
SQL リファレンス 第 2 巻	SA88-4675-01	入手可能	2013 年 1 月
Text Search ガイド	SA88-4692-01	入手可能	2013 年 1 月
問題判別およびデータベース・パフォーマンスのチューニング	SA88-4664-01	入手可能	2013 年 1 月
DB2 バージョン 10.1 へのアップグレード	SA88-4678-01	入手可能	2013 年 1 月
DB2 バージョン 10.1 の新機能	SA88-4684-01	入手可能	2013 年 1 月
XQuery リファレンス	SA88-4687-01	入手不可	2013 年 1 月

表 34. DB2 Connect 固有の技術情報

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
DB2 Connect Personal Edition インストールおよび構成	SA88-4681-00	入手可能	2012 年 4 月
DB2 Connect サーバー機能 インストールおよび構成	SA88-4682-01	入手可能	2013 年 1 月
DB2 Connect ユーザーズ・ガイド	SA88-4683-01	入手可能	2013 年 1 月

コマンド行プロセッサーから SQL 状態ヘルプを表示する

DB2 製品は、SQL ステートメントの結果として生じる可能性がある状態に対応した SQLSTATE 値を戻します。 SQLSTATE ヘルプは、SQL 状態および SQL 状態クラス・コードの意味を説明します。

手順

SQL 状態ヘルプを開始するには、コマンド行プロセッサーを開いて以下のように入力します。

```
? sqlstate または ? class code
```

ここで、*sqlstate* は有効な 5 桁の SQL 状態を、*class code* は SQL 状態の最初の 2 桁を表します。
例えば、`? 08003` を指定すると SQL 状態 08003 のヘルプが表示され、 `? 08` を指定するとクラス・コード 08 のヘルプが表示されます。

異なるバージョンの DB2 インフォメーション・センターへのアクセス

他のバージョンの DB2 製品の資料は、ibm.com® のそれぞれのインフォメーション・センターにあります。

このタスクについて

DB2 バージョン 10.1 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v10r1 です。

DB2 バージョン 9.8 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9r8/ です。

DB2 バージョン 9.7 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/ です。

DB2 バージョン 9.5 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9r5 です。

DB2 バージョン 9.1 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9/ です。

DB2 バージョン 8 のトピックについては、*DB2 インフォメーション・センター* の URL (http://publib.boulder.ibm.com/infocenter/db2luw/v8/) を参照してください。

コンピューターまたはイントラネット・サーバーにインストールされた DB2 インフォメーション・センターの更新

ローカルにインストールした DB2 インフォメーション・センターは、定期的に更新する必要があります。

始める前に

DB2 バージョン 10.1 インフォメーション・センターが既にインストール済みである必要があります。詳しくは、「*DB2 サーバー機能 インストール*」の『DB2 セットアップ・ウィザードによる DB2 インフォメーション・センターのインストール』のトピックを参照してください。インフォメーション・センターのインストールに適用されるすべての前提条件と制約事項は、インフォメーション・センターの更新にも適用されます。

このタスクについて

既存の DB2 インフォメーション・センターは、自動で更新することも手動で更新することもできます。

- 自動更新は、既存のインフォメーション・センターのフィーチャーと言語を更新します。自動更新を使用すると、手動更新と比べて、更新中にインフォメーション・センターが使用できなくなる時間が短くなるというメリットがあります。さらに、自動更新は、定期的に実行する他のバッチ・ジョブの一部として実行されるように設定することができます。
- 手動更新は、既存のインフォメーション・センターのフィーチャーと言語の更新に使用できます。自動更新は更新処理中のダウン時間を減らすことができますが、フィーチャーまたは言語を追加する場合は手動処理を使用する必要があります。例えば、ローカルのインフォメーション・センターが最初は英語とフランス語でインストールされており、その後ドイツ語もインストールすることにした場合、手動更新でドイツ語をインストールし、同時に、既存のインフォメーション・センターのフィーチャーおよび言語を更新できます。しかし、手動更新ではインフォメーション・センターを手動で停止、更新、再始動する必要があります。更新処理の間はずっと、インフォメーション・センターは使用できなくなります。自動更新処理では、インフォメーション・センターは、更新を行った後に、インフォメーション・センターを再始動するための停止が発生するだけで済みます。

このトピックでは、自動更新のプロセスを詳しく説明しています。手動更新の手順については、『コンピューターまたはイントラネット・サーバーにインストールされた DB2 インフォメーション・センターの手動更新』のトピックを参照してください。

手順

コンピューターまたはイントラネット・サーバーにインストールされている DB2 インフォメーション・センターを自動更新する手順を以下に示します。

1. Linux オペレーティング・システムの場合、次のようにします。
   a. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、DB2 インフォメーション・センターは、`/opt/ibm/db2ic/V10.1` ディレクトリーにインストールされています。
   b. インストール・ディレクトリーから `doc/bin` ディレクトリーにナビゲートします。
   c. 次のように `update-ic` スクリプトを実行します。

      ```
      update-ic
      ```
2. Windows オペレーティング・システムの場合、次のようにします。
   a. コマンド・ウィンドウを開きます。
   b. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、DB2 インフォメーション・センターは、`<Program Files>¥IBM¥DB2 Information Center¥バージョン 10.1` ディレクトリーにインストールされています (`<Program Files>` は「Program Files」ディレクトリーのロケーション)。

c. インストール・ディレクトリーから doc¥bin ディレクトリーにナビゲートします。

d. 次のように update-ic.bat ファイルを実行します。

```
update-ic.bat
```

タスクの結果

DB2 インフォメーション・センターが自動的に再始動します。更新が入手可能な場合、インフォメーション・センターに、更新された新しいトピックが表示されます。インフォメーション・センターの更新が入手可能でなかった場合、メッセージがログに追加されます。ログ・ファイルは、doc¥eclipse¥configuration ディレクトリーにあります。ログ・ファイル名はランダムに生成された名前です。例えば、1239053440785.log のようになります。

コンピューターまたはイントラネット・サーバーにインストールされた DB2 インフォメーション・センターの手動更新

DB2 インフォメーション・センターをローカルにインストールしている場合は、IBM から資料の更新を入手してインストールすることができます。

このタスクについて

ローカルにインストールされた *DB2 インフォメーション・センター* を手動で更新するには、以下のことを行う必要があります。

1. コンピューター上の *DB2 インフォメーション・センター* を停止し、インフォメーション・センターをスタンドアロン・モードで再始動します。インフォメーション・センターをスタンドアロン・モードで実行すると、ネットワーク上の他のユーザーがそのインフォメーション・センターにアクセスできなくなります。これで、更新を適用できるようになります。DB2 インフォメーション・センターのワークステーション・バージョンは、常にスタンドアロン・モードで実行されます。を参照してください。
2. 「更新」機能を使用することにより、どんな更新が利用できるかを確認します。インストールしなければならない更新がある場合は、「更新」機能を使用してそれを入手およびインストールできます。

   注: ご使用の環境において、インターネットに接続されていないマシンに *DB2 インフォメーション・センター* の更新をインストールする必要がある場合、インターネットに接続されていて *DB2 インフォメーション・センター* がインストールされているマシンを使用して、更新サイトをローカル・ファイル・システムにミラーリングしてください。ネットワーク上の多数のユーザーが資料の更新をインストールする場合にも、更新サイトをローカルにミラーリングして、更新サイト用のプロキシーを作成することにより、個々のユーザーが更新を実行するのに要する時間を短縮できます。
   更新パッケージが入手可能な場合、「更新」機能を使用してパッケージを入手します。ただし、「更新」機能は、スタンドアロン・モードでのみ使用できます。
3. スタンドアロンのインフォメーション・センターを停止し、コンピューター上の *DB2 インフォメーション・センター* を再開します。

注: Windows 2008、Windows Vista (およびそれ以上) では、このセクションの後の部分でリストされているコマンドは管理者として実行する必要があります。完全な管理者特権でコマンド・プロンプトまたはグラフィカル・ツールを開くには、ショートカットを右クリックしてから、「**管理者として実行**」を選択します。

手順

コンピューターまたはイントラネット・サーバーにインストール済みの *DB2* インフォメーション・センター を更新するには、以下のようにします。

1. *DB2* インフォメーション・センター を停止します。
   - Windows では、「**スタート**」 > 「**コントロール パネル**」 > 「**管理ツール**」 > 「**サービス**」をクリックします。次に、「**DB2 インフォメーション・センター**」サービスを右クリックして「**停止**」を選択します。
   - Linux では、以下のコマンドを入力します。

     ```
     /etc/init.d/db2icdv10 stop
     ```
2. インフォメーション・センターをスタンドアロン・モードで開始します。
   - Windows の場合:
     a. コマンド・ウィンドウを開きます。
     b. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、*DB2* インフォメーション・センター は、*Program_Files*¥IBM¥DB2 Information Center¥バージョン 10.1 ディレクトリーにインストールされています (*Program_Files* は Program Files ディレクトリーのロケーション)。
     c. インストール・ディレクトリーから doc¥bin ディレクトリーにナビゲートします。
     d. 次のように help_start.bat ファイルを実行します。

        ```
        help_start.bat
        ```
   - Linux の場合:
     a. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、*DB2* インフォメーション・センター は、/opt/ibm/db2ic/V10.1 ディレクトリーにインストールされています。
     b. インストール・ディレクトリーから doc/bin ディレクトリーにナビゲートします。
     c. 次のように help_start スクリプトを実行します。

        ```
        help_start
        ```

   システムのデフォルト Web ブラウザーが開き、スタンドアロンのインフォメーション・センターが表示されます。
3. 「**更新**」ボタン () をクリックします。(ブラウザーで JavaScript が有効になっている必要があります。) インフォメーション・センターの右側のパネルで、「**更新の検索**」をクリックします。 既存の文書に対する更新のリストが表示されます。
4. インストール・プロセスを開始するには、インストールする更新をチェックして選択し、「**更新のインストール**」をクリックします。
5. インストール・プロセスが完了したら、「**完了**」をクリックします。

6. 次のようにして、スタンドアロンのインフォメーション・センターを停止します。
   - Windows の場合は、インストール・ディレクトリーの doc¥bin ディレクトリーにナビゲートしてから、次のように help_end.bat ファイルを実行します。

     ```
     help_end.bat
     ```

     注: help_end バッチ・ファイルには、help_start バッチ・ファイルを使用して開始したプロセスを安全に停止するのに必要なコマンドが含まれています。help_start.bat は、Ctrl-C や他の方法を使用して停止しないでください。
   - Linux の場合は、インストール・ディレクトリーの doc/bin ディレクトリーにナビゲートしてから、次のように help_end スクリプトを実行します。

     ```
     help_end
     ```

     注: help_end スクリプトには、help_start スクリプトを使用して開始したプロセスを安全に停止するのに必要なコマンドが含まれています。他の方法を使用して、help_start スクリプトを停止しないでください。
7. *DB2 インフォメーション・センター* を再開します。
   - Windows では、「**スタート**」 > 「**コントロール パネル**」 > 「**管理ツール**」 > 「**サービス**」をクリックします。次に、「**DB2 インフォメーション・センター**」サービスを右クリックして「**開始**」を選択します。
   - Linux では、以下のコマンドを入力します。

     ```
     /etc/init.d/db2icdv10 start
     ```

タスクの結果

更新された *DB2 インフォメーション・センター* に、更新された新しいトピックが表示されます。

DB2 チュートリアル

DB2 チュートリアルは、DB2 データベース製品のさまざまな機能について学習するための支援となります。この演習をとおして段階的に学習することができます。

はじめに

インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/) から、このチュートリアルの XHTML 版を表示できます。

演習の中で、サンプル・データまたはサンプル・コードを使用する場合があります。個々のタスクの前提条件については、チュートリアルを参照してください。

DB2 チュートリアル

チュートリアルを表示するには、タイトルをクリックします。

「*pureXML ガイド* 」の『**pureXML®**』
: XML データを保管し、ネイティブ XML データ・ストアに対して基本的な操作を実行できるように、DB2 データベースをセットアップします。

DB2 トラブルシューティング情報

DB2 データベース製品を使用する際に役立つ、トラブルシューティングおよび問題判別に関する広範囲な情報を利用できます。

DB2 の資料

トラブルシューティング情報は、「*問題判別およびデータベース・パフォーマンスのチューニング*」または *DB2 インフォメーション・センター* の『データベースの基本』セクションにあります。ここには、以下の情報が記載されています。

- DB2 診断ツールおよびユーティリティーを使用した、問題の切り分け方法および識別方法に関する情報。
- 最も一般的な問題のうち、いくつかの解決方法。
- DB2 データベース製品で発生する可能性のある、その他の問題の解決に役立つアドバイス。

IBM サポート・ポータル

現在問題が発生していて、考えられる原因とソリューションを見つけるには、IBM サポート・ポータルを参照してください。 Technical Support サイトには、最新の DB2 資料、TechNotes、プログラム診断依頼書 (APAR またはバグ修正)、フィックスパック、およびその他のリソースへのリンクが用意されています。この知識ベースを活用して、問題に対する有効なソリューションを探し出すことができます。

IBM サポート・ポータル (http://www.ibm.com/support/entry/portal/Overview/Software/Information_Management/DB2_for_Linux,_UNIX_and_Windows) にアクセスしてください。

ご利用条件

これらの資料は、以下の条件に同意していただける場合に限りご使用いただけます。

適用度: これらのご利用条件は、IBM Web サイトのあらゆるご利用条件に追加で適用されるものです。

個人使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

商業的使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。 ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

権利: ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態で提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されます。

IBM の商標: IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corporation の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

付録 F. 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。IBM 以外の製品に関する情報は、本書の最初の発行時点で入手可能な情報に基づいており、変更される場合があります。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒103-8510
東京都中央区日本橋箱崎町19番21号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Canada Limited
U59/3600
3600 Steeles Avenue East
Markham, Ontario L3R 9Z7
CANADA

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、

利便性もしくは機能性があることをほのめかしたり、保証することはできません。サンプル・プログラムは、現存するままの状態で提供されるものであり、いかなる種類の保証も提供されません。 IBM は、これらのサンプル・プログラムの使用から生ずるいかなる損害に対しても責任を負いません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生した創作物には、次のように、著作権表示を入れていただく必要があります。

© (*お客様の会社名*) (*西暦年*). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 © Copyright IBM Corp. _*年を入れる*_. All rights reserved.

商標

IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corporation の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

以下は、それぞれ各社の商標または登録商標です。

- Linux は、Linus Torvalds の米国およびその他の国における商標です。
- Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。
- UNIX は The Open Group の米国およびその他の国における登録商標です。
- インテル、Intel、Intel ロゴ、Intel Inside、Intel Inside ロゴ、Celeron、Intel SpeedStep、Itanium、Pentium は、Intel Corporation または子会社の米国およびその他の国における商標または登録商標です。
- Microsoft、 Windows、 Windows NT、および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

A

C

D

F

G

I

L

M

P

R

S

U

X

Printed in Japan

SA88-4693-01

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21

Spine information:

IBM DB2 10.1 for Linux, UNIX, and Windows

データ移動ユーティリティー: ガイドおよびリファレンス